

**A Java based application to check and report
the integrity of links to resources in a web site**

BSc (Hons) Computing Science
Staffordshire University

A project submitted in partial fulfilment of the award of the
degree of BSc (Hons) Computing Science from Staffordshire
University

Supervised by Tracy Lewis

May 2001

CONTENTS

Abstract.....	v
CHAPTER 1: INTRODUCTION	0
1 Introduction	1
1.1 Background.....	1
1.2 Objectives	1
CHAPTER 2: PROJECT DELIVERABLES	1
2 Project deliverables.....	2
2.1 Research.....	2
2.2 Analysis	2
2.3 Design and implementation	2
2.4 Project management	3
CHAPTER 3: RESEARCH.....	3
3 Similar products.....	4
3.1 Xenu’s link sleuth	4
3.1.1 Good points.....	4
3.1.2 Bad points	5
3.1.3 Conclusion	5
3.2 Link police	5
3.2.1 Good points.....	6
3.2.2 Bad points	6
3.2.3 Conclusion	6
3.3 Netmechanic	7
3.3.1 Good points.....	7
3.3.2 Bad points	7
3.3.3 Conclusion	7
3.4 Anchor Checker	8
3.4.1 Good points.....	8
3.4.2 Bad points	8
3.4.3 Conclusion	9
3.5 Results	9
4 Research into HCI.....	10
4.1 Colour.....	10
4.2 Screen layout.....	12
4.3 Usability.....	12
4.3.1 Familiarisation.....	14

4.3.2	Memorisation	14
4.3.3	Errors	14
4.3.4	Efficiency	15
4.3.5	Satisfaction.....	15
5	The Robot Exclusion Standard.....	16
5.1	Definition	16
5.2	Implementation	17
6	The HTML	18
6.1	Tags	18
6.2	Attributes	19
6.2.1	A tag	19
6.2.2	APPLET tag.....	21
6.2.3	AREA tag.....	21
6.2.4	BASE tag	21
6.2.5	BLOCKQUOTE tag.....	22
6.2.6	BODY tag	22
6.2.7	FORM tag	22
6.2.8	HEAD tag.....	23
6.2.9	IMG tag.....	23
6.2.10	INPUT tag.....	23
6.2.11	LINK tag.....	23
6.2.12	SCRIPT tag.....	24
	CHAPTER 4: ANALYSIS.....	24
7	Problems and solutions encountered during research.....	25
7.1	HCI	25
7.2	Similar products	25
7.3	Robot Exclusion Standard	26
7.4	HTML tags.....	26
7.5	Summary.....	26
8	Programming language analysis.....	27
8.1	Java and C.....	27
8.1.1	Java	27
8.1.2	C	28
8.2	Comparison of Java and C.....	28
8.2.1	Primitive data types comparison	29
8.2.2	Operator precedence comparison	30
8.2.3	Control statements comparison	32
8.3	Imports, includes and other differences.....	32
8.4	Which language will it be?	34
9	Functional requirements.....	34

9.1	Network connections.....	34
9.2	Robot Exclusion Standard	34
9.3	Data retrieval.....	35
9.4	Data parsing	35
9.5	Link history.....	35
9.6	Depth	36
9.7	Graphical user interface.....	36
9.8	Reporting findings.....	37
9.9	Other features.....	37
10	Design method.....	38
10.1	Jackson system development	38
10.1.1	The modeling stage.....	38
10.1.2	The network stage	39
10.1.3	The implementation stage.....	40
10.2	The UML	40
10.2.1	Class diagram.....	41
10.2.2	Object diagram.....	41
10.2.3	State diagram.....	42
10.2.4	Use Case diagram.....	43
10.2.5	Sequence diagram.....	43
10.2.6	Activity diagram.....	44
10.2.7	Collaboration diagram	45
10.2.8	Summary.....	45
10.3	JSD or the UML?	46
	CHAPTER 5: DESIGN & IMPLEMENTATION.....	46
11	Testing and evaluation.....	47
11.1	Testing	47
11.2	Evaluation.....	47
12	Hardware.....	47
13	Design and implementation	49
13.1	Tools.....	49
13.1.1	JDK 1.3 and notepad	49
13.1.2	JDK 1.3 and ultra-edit	49
13.1.3	Jbuilder 4 foundation.....	50
13.1.4	Together control center.....	50
13.1.5	Forté for Java community edition	51
13.1.6	Summary.....	51
13.2	Diagrams.....	51
13.2.1	Use case diagram.....	51

13.2.2 Class design.....	52
13.2.3 Sequence diagrams	67
13.2.4 User interface	75
13.2.5 User manual	87
CHAPTER 6: TESTING	87
14 Testing88	
14.1 Functionality tests	88
14.1.1 Results.....	89
14.2 Comparison test.....	92
14.2.1 Comparison test result	92
14.3 Invalid entry test.....	92
14.3.1 Invalid entry test results.....	92
CHAPTER 7: EVALUATION	93
15 Evaluation	94
15.1 Expert evaluation	94
15.1.1 Heuristic evaluation.....	94
15.2 Evaluation with the user	97
15.2.1 User characteristics.....	97
15.2.2 Interface evaluation with the user	99
15.2.3 User satisfaction questionnaire	101
15.3 Critical evaluation	103
15.3.1 Problems encountered during this project	104
15.3.2 Lessons learnt	104
15.3.3 Things I would have done differently	105
15.3.4 Conclusion	105
CHAPTER 8: REFERENCES	105
APPENDIX A: USER MANUAL.....	I
APPENDIX B: SOURCE CODE	VIII
APPENDIX C: GANNT CHART AND WEEKLY LOGBOOK.....	CXLVII

Abstract

As Internet sites increase in size and complexity, the action of maintaining them becomes more and more difficult to manage. To continually check the status of a very large website manually is no longer possible. To solve this problem, a software robot is used to validate the integrity of the many linked resources, which form a modern site. This report describes the design and implementation of such a system.

CHAPTER 1

INTRODUCTION

1 Introduction

This report focuses on the design and development of a stand-alone web agent application which automates the maintenance of a website. Already existing products provide the initial research, with the goal of designing and implementing a system that brings something new and useful to website designers and developers.

1.1 Background

The problem of maintaining a web site is well known in the Internet world. Existing systems provide mechanisms to help web site administrators make sure that their web site is up to date and that all links pointing to resources are valid. Unfortunately the existing programs available on the market lack some features that could be of importance, I have therefore decided to investigate those existing applications and create a comparable software with additional functionality.

1.2 Objectives

The objectives I would like to achieve in this project fall into three categories. The first is to do some research into similar products, the HTML language, the robot exclusion standard and HCI (Human Computer Interaction).

The second objective is to design and implement a system that will allow a user to scan a web site and check for the integrity of resources within it.

The third is to produce a completed report documenting the different stages of the project. This will include analysis, design and implementation, to finish with testing and evaluation. Appendices will contain a user manual, code listing, and the project's logbook including a Gantt chart describing the time management of this project.

CHAPTER 2

PROJECT

DELIVERABLES

2 Project deliverables

The deliverables for this project fall into four main sections, they are as follow:

- Research
- Analysis
- Design and implementation
- Project management

2.1 Research

This is a major section within this project, it will include research into HCI, HTML, robot exclusion standard and similar software to the one that will be built. Research will allow me to start the analysis section with enough knowledge and understanding of what has to be achieved.

2.2 Analysis

This section will analyse what functionality the software should have, what programming language should be used to implement it, and solve any possible problems found during the research section. This section will also include a discussion on what methodology to use for designing and implementing the system.

2.3 Design and implementation

The design section will contain screen designs as well as the core design of the software and the way it is implemented. Testing and evaluation will be included at the end of the report.

2.4 Project management

The project management stage of the project is there to control the different resources and processes needed within a given time allocation. A Gantt chart can be found in appendix C which describes the time allocation of each processes.

CHAPTER 3

RESEARCH

3 Similar products

There are a variety of software solutions available for simplifying website maintenance. Several different approaches are examined here to compare different features, and gain an understanding of the best approach to take when designing a new web robot. This list is not exhaustive, on the contrary, many robots already exist out there, but most if not all of them are missing what is thought to be basic functions.

In the following pages, good and bad points about each surveyed applications will be examined and understanding gained on what to add or improve in the software to be.

The layout of this examination is as follow:

- The name of the application being surveyed.
- A list of good points about the application.
- A list of bad points or improvement needed.
- A small conclusion and what problems arise when using the software

3.1 Xenu's link sleuth

Xenu's Link Sleuth is easy to utilise. It provides the user with a comprehensive list of options, and uses multithreading in its link-checking algorithm. The amount of threads used can be set at will.

3.1.1 Good points

- Fast
- Multi threaded.
- HTML report
- Free software
- Stand alone

3.1.2 Bad points

- No email report
- HTML report is basic
- Does not obey the robot exclusion standard
- Gives unwanted information (unneeded) such as file size, date and time, title of link, server the link is on.

3.1.3 Conclusion

A few problems arise when using Link Sleuth, the main one being the report in the HTML format. The results displayed are difficult to understand due to poor document layout. The information displayed is also very varied and may be too heavy for most users. The broken links found by Link Sleuth are displayed without their originating page or parent page, which means that it is practically impossible to know which page needs to be repaired, and this in itself makes what seems to be a good package a software to avoid.

Overall, this application could become very useful indeed with only a few changes, it is fast and seems to be reliable in the sense that it has not missed any broken links when it was tried for this research.

The Xenu's link sleuth can be found at <http://home.snafu.de/tilman/xenulink.html>.

3.2 Link police

Link Police is a web-based software; it runs on a server and reports its findings via email. To use it, the user must sign up online and pay an expensive fee yearly. The user has no access to any options what so ever. A demonstration was used to check a site well known to the author, with great success. The software sent results to an email address within 2 minutes. It contained a detailed report of broken links within the site.

3.2.1 Good points

- E-mail report
- Seems fast
- Suspect it is multi threaded
- Web imbedded software

3.2.2 Bad points

- No HTML report
- Expensive
- Only checks images and http links
- Do not know if robot exclusion standard is being implemented

3.2.3 Conclusion

Unfortunately, the scan seems to concentrate only on links to other pages and on image resources, other resources such as applets or file descriptors are not picked up. This lack of functionality is rather disturbing, especially when the price tag is so high.

The report was well laid out, with the broken links names and parent pages displayed adequately.

Overall, this web-based service could be improved greatly by adding the possibility for the user to choose the way the scans are done, via a set of options. It should also allow checking for other resources but images and page links. A price reduction would also be welcomed, or the license could be set to a life one instead of a yearly one.

The Link police can be found at <http://linkpolice.mycomputer.com>

3.3 Netmechanic

Netmechanic is another web-based software. It runs server side. It is a suite of different web site developer's tools. The link-checking tool is actually called HTML toolbox.

3.3.1 Good points

- Online report
- Web imbedded software
- Multi threaded
- Repair pages

3.3.2 Bad points

- Slow
- No email reporting
- Do not know if the robot exclusion standard is implemented
- Annual fee

3.3.3 Conclusion

It seems very slow compare to the other software, but it does something that is rarely done among web robots, it actually fixes most of the problems encountered. It obviously cannot fix broken links, but when it encounters code errors, it attempts to repair them. The idea seems good at first glance, but looking at it with the designer's eyes, it seems to be a dangerous thing to do. If an error is encountered, is it really an error or did the web designer intend to code the page that way.

Reporting is done online in a HTML fashion; it displays only a count of broken links and lists them. The report layout is cumbersome and does not really give any interesting information. It reports things such as time taken to connect to resources, which is totally unneeded.

A few options are available to the user, which makes it a little more attractive than the Link Police software.

Netmechanic can be found at <http://www.netmechanic.com/>

3.4 Anchor Checker

You can use regular expression to specify your files, one example: checker *.html. Giving it some options can control the behaviour. The program only checks anchor tags, it is very efficient in doing so and very fast, but it is also very limited, not even images are being checked.

3.4.1 Good points

- Stand alone software
- Multi threaded
- Fast
- Reliable

3.4.2 Bad points

- Difficult to use
- No email or HTML reporting
- Command line usage
- Needs compiler to work initially

3.4.3 Conclusion

It is a freely distributed application. Novice users cannot really use it as it has to be compiled first, and all the options are being passed to the software at the command line. It is originally a UNIX program for UNIX users, although it can be compiled on a Windows machine. The reporting is in text format; there is no way to send a report via email or to create one in the TML format.

Overall, this product is poorly presented, and lacks in friendliness. Although it is freely available, it would not make a good tool to a web developer.

Anchor Checker can be found at: <http://www.abdn.ac.uk/tools/unix/checker/>

3.5 Results

The goal of this research is to examine some of the best features of existing systems and come up with characteristics for the new application that offer the user something different to current link checkers.

The principle failing of the aforementioned systems is a poor reporting facility and an overly complicated display or no display at all (command line). A display easier to understand and manage, better reporting technique and some original features should add an improvement on current solutions.

4 Research into HCI

Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major facts surrounding them.

4.1 Colour

The human eye contains cones and rods, which are two different types of light receptors. The cones are the ones we are the most interested in the context of this report.

There are three kinds of cones, all sensitive to colour, these colours being red, green and blue (or very close to those colours). They are not all sensitive to the same level; for example, the green cones are the most sensitive, as the blue ones are the less sensitive to light. This means that different colours are better suited for different tasks.

Over the years, researchers all over the world have discovered that close to 8% of the male population has some degree of colour blindness or colour impairment. It mostly translates into an incapability to differentiate between green and red.

These important facts must be remembered when one attempts to design a user interface, and colour should not be the only thing a software designer relies on when creating a display containing some level of colour coding. Other techniques must also be applied, such as symbols, shapes and sizes of interface components. People are very good at perceiving patterns, or structures. The best every day example of colour coding and layout is in traffic lights. The positions of the lights are the same all over the world and it would surely be a disaster if traffic lights were to have only one lamp that would change colour. This arrangement means that people do not have to rely on being able to differentiate between colours.

Some combinations of colour are strongly inadvisable, red and blue are the best example in this case. When light goes through the eye, it is bent by different amount depending on its wavelength; this makes it very hard for the eye to focus on all the colours at the same time. The phenomenon that arises when using ill-assorted colours is called a chromatic aberration. Looking at the examples on the next page, one can see that it can be extremely difficult to read the text, but most

importantly, the strain on the eye is multiplied greatly and it becomes impossible after only a small amount of time to look at some of these colour combinations.

Red on blue	Red on blue makes text appear to 'vibrate'
Yellow on blue	Yellow on blue makes the edges around the text look pale.
Red on green	This combination gives a shadow effect
Green on blue	This may create an 'afterimage' on the retina which could impede vision for a short period of time.

Figure 1: Colour mixing

Using colour in user interfaces has become an everyday occurrence, it can make the use of the software via its interface more efficient, and it also provides a more aesthetically pleasing interface to the user. Most scientists studying this area of human computer interaction and most software designers agree that a display should be designed as if it was going to be monochrome, then colour added suitably to improve the interface.

Research indicates that our memory for colour-highlighted elements is better than for monochrome. Interface components of different intensity, (brightness or lightness) but of similar hue help to draw the user's attention or to focus on particular elements on the screen. In practical terms, it is not easy for people to differentiate reliably between more than two levels of brightness. It is easier if the two elements are close together on the screen, but more complicated if they are far away from each other.

When different intensities are used to distinguish between software components, it is very important to make sure that the difference is significant, if not, it could have the opposite effect and get the user to skip over certain elements.

There are general guidelines when it comes to using colours in interface design; here are a few of them:

- Try not to use more than 4 to 5 colours on the same screen.
- A colour code should uphold the user's task, not hamper it.
- Colour consistency must be kept among software interfaces.

If possible, the user should be able to control the colour coding so that he/she can assign colours that have some sense for them.

4.2 Screen layout

There are basic differences that should be recognized when choosing to use a computer screen over a sheet of paper to convey information. For example, a designer laying out a page in a desktop publishing application will have the knowledge that the area he/she has to work with is of a set size, usually A4 (210mm by 297mm). Yet, a computer screen has no set dimensions. While there are standard resolutions like VGA (640 x 480) and SVGA (800 x 600, 1024 x 768, etc.), a user may not have its application window fully opened. So how does software designers know what size has to be dealt with? The most probable answer is that they do not know in advance what resolution a screen is set to and how a user prefers to use its application windows, they usually have to use certain programming techniques to insure that the layout of the windows on the screen is suitable for the application.

The different components making the user interface should be placed logically. All the elements must have a visual association with at least one other, it could be colour coded or shape oriented, and the components can and should be placed in groups representing the different drop down menu items usually available in an interface.

4.3 Usability

This refers to how much effort has to be put in by the user to run the software. Obviously for a good piece of software, the user should put as little effort in as possible. The best software packages have as little user interaction as possible and thus usability becomes much greater. Usability encompasses different aspect of

user systems, such as familiarisation, memorisation, software error handling, application efficiency, etc...

On the next pages are short discussions on some of these aspects.

4.3.1 Familiarisation

An application should be easy to learn so that the user can quickly begin undertaking the work to be done. This quality is closely joined with memorisation since normally what is easily learnt is easily remembered too. System navigation will definitely play a significant role in getting to know the application. The more complex the software is, the harder it will be to learn and use.

4.3.2 Memorisation

An application should be simple to memorise, so the user can go back to it after a break without having to re-learn large areas of it. A solution to creating unforgettable software is to make all the window layouts consistent. That is uniformity in terms of the location of menu items, components used, colours in use, etc. “Consistency is a hallmark of good instructional design; if items are consistent throughout instruction, then the learner can devote more energy to dealing with the content of a presentation than to learning (and re-learning) the conventions of the delivery system.” (Misanchuk & Schwier, 1995).

4.3.3 Errors

An application should contain few errors if any. It should also permit users to recover from errors effortlessly. It is possible for a user to make what is called software errors when using menu controls and buttons. As a result, it is important to make sure that these interactive parts of the software can handle not only predicted data but also illogical figures. If an error has been made then the software should show the user feedback stating the following:

- An error has just occurred.
- An explanation of why it occurred.
- What action should be undertaken to correct the error and avoid repeating it in the future.

4.3.4 Efficiency

An application should sustain a high level of production. Therefore, cautious thoughts should be given to the reason of the software layout. One way of supporting a high level of production is to clearly label all the components in the user interface (speak the users language). The main reason for doing this is to provide an easy to use and intuitive software interface. This means that when new users come into contact with the application for the first time, they can easily see the different things the application can do and rapidly make a judgment on what to do with it.

4.3.5 Satisfaction

An application should be pleasing to use, so that users are intuitively contented when using it. Satisfaction is the most indescribable of the usability attributes. It is not easy to design for because what is subjectively pleasing could be infuriating for a different user. Over whole, if an application is easily learnt, supports a high level of efficiency, is memorable, and can without difficulty recover from errors then the users should already be contented. On the other hand, there are a few other points that should be well thought-out. For instance, the utilisation of colours can make an application more satisfying than a monochrome one. Easy navigation through the application is very important too and plays a great role in user satisfaction. In the end, every user cannot be pleased as everyone sees and feels in different ways, but good research should be carried out before any interface is to be built.

5 The Robot Exclusion Standard

Many servers might consider automated clients or robots such as the application being developed, an invasion of resources. A robot is defined as a web client that may retrieve documents in a mechanized, rapid-fire sequence. For example, some robots are link traversal programs, indexers for search engines, or content mirroring applications. While many webmasters greet robots, others prefer them to avoid their servers and stay out.

5.1 Definition

The Robot Exclusion Standard was devised in 1994 to give web site administrators the prospect of making their preferences known to those robots. It explains how a web server administrator can select certain areas of a site as "out of bounds" for certain or all web clients. The success of the Robot Exclusion Standard depends on web robot programmers being thorough and implementing it carefully. It can be seen as a sign like "Do Not Disturb". It can be ignored at the robot's user's own risk. Persisting in using a robot which does not obey the standard can bring complaints to the user and can also have the server's administrator permanently lock out the IP address or entire domain name from which the offence came from. This in turn can lead to serious problems such as a job loss if the robot was used from a worker's office or from a company's network.

In a nutshell, the Robot Exclusion Standard states that a Webmaster should create a file accessible at the relative URL */robots.txt*. For example, a remote client would access a *robots.txt* file at the server *www.javaspy.net* using the following URL: <http://www.javaspy.net/robots.txt>

If the server returns a response code of 200 (OK) for the URL, the application should download the file for parsing and interpretation. In other cases, response codes in the range of 300-399 indicate redirections, which should be followed by the robot. Response codes of 401 (Unauthorized) or 403 (Forbidden) indicate restrictions and the client should avoid the entire site. A 404 (Not Found) response code means that the administrator did not indicate any Robot Exclusion Standard and the entire site is okay to be visited by the client.

On the next page is a detailed explanation of how the standard is implemented.

5.2 Implementation

When clients receive the *robots.txt* file, they need to parse it to determine whether they are allowed access to the site. There are three basic directives that can be in the *robots.txt* file:

- User-agent
- Allow
- Disallow

The User-agent directive specifies that subsequent Allow and Disallow statements apply to it. The robot should use a case-insensitive comparison of this value with its own user agent name. Version numbers are not used in the comparison.

If the robots.txt file specifies a * as a User-Agent, it indicates all robots, not any particular robot. So if an administrator wants to shut out all robots from an entire site, the robots.txt file only needs the following two lines:

- User-agent: *
- Disallow: /

The Allow and Disallow directives indicate areas of the site that the previously listed user-agent is allowed or denied access. Instead of listing all the URLs that the user-agent is allowed and disallowed, the directive specifies the general prefix that describes what is allowed or disallowed. For example:

- Disallow: /index

would match both /index.html and /index/summary.html, while:

- Disallow: /index/

would match only URLs in /index/. In the extreme case, Disallow: /

specifies the entire web site.

Multiple user-agents can be specified within a *robots.txt* file. For example:

- User-agent: friendly-indexer
- User-agent: search-thingy

- Disallow: /cgi-bin/
- Allow: /

specifies that the allow and disallow statements apply to both the friendly-indexer and search-thingy robots.

The *robots.txt* file moves from general to specific; that is, subsequent listings can override previous ones. For example:

- User-agent: *
- Disallow: /
- User-agent: search-thingy
- Allow: /

would specify that all robots should go away, except the *search-thingy* robot.

6 The HTML

The hypertext mark-up language (HTML) has been around a while now, version 4 is the latest version to be used, but most browsers including the most common ones do not implement it yet. It is felt by the author that by the time this report comes out and by the time the software developed in parallel is finished, version 4 of HTML will still not be widely used across the Internet community. Therefore version 3.2.2 will be used in the implementation phase of the application's development.

6.1 Tags

This part of the research discusses the HTML tags that are used to point to resources across the World Wide Web or are used by other tags to understand where a resource may be located. One important thing to be considered is the compatibility between browsers, especially the most commonly used ones such as Netscape® Navigator and Microsoft® Internet Explorer. If during this research it is found that a HTML tag representing a resource can only be understood by

Navigator or IE, it will be pointed out, it will not be used in the implementation of the software as this means that it is not HTML 3.2.2 compliant.

Below is a table of all the tags used in the implementation of the HTML version 3.2.2.that can point to resources on the World Wide Web.

It has to be said that the tags themselves do not point to resources, rather one or more of the tags attributes.

Resource Tag	HTML 3.2.2	Netscape	I.E	To be used
A	Yes	Yes	Yes	Yes
APPLET	Yes	Yes	Yes	Yes
AREA	Yes	Yes	Yes	Yes
BASE	Yes	Yes	Yes	Yes
BLOCKQUOTE	Yes	Yes	Yes	Yes
BODY	Yes	Yes	Yes	Yes
FORM	Yes	Yes	Yes	Yes
HEAD	Yes	Yes	Yes	Yes
IMG	Yes	Yes	Yes	Yes
INPUT	Yes	Yes	Yes	Yes
LINK	Yes	Yes	Yes	Yes
SCRIPT	Yes	Yes	Yes	Yes

Figure 2: Tags

6.2 Attributes

This section describes in details the fourteen resource tags and their attributes.

6.2.1 A tag

This tag represents a connection from one web resource to another. It is used as an anchor to mark the beginning and/or the end of a hypertext link.

It has many different attributes, but the ones needed for this application are the href attribute and the name attribute.

- The href attribute declares the supplied URL (Uniformed Resource Locator) to be the *target* of this anchor, i.e. the resource that will be retrieved if the user clicks on it.

- The name attribute declares the anchor to be available as a *target* for links. When used as the href value in an anchor, the browser places this anchor near to the top of the window.

6.2.2 APPLETTAG

This tag is used to embed a Java applet into the document. A Java applet is a program, written in the Java language. The browser assigns a rectangle portion of the window to the applet in which it runs. The size of this region is set in the HTML page.

The attributes code and codebase will be implemented in the software.

- The code attribute gives the name of the file containing the *sub-class* of the compiled applet or the path pointing to the *class* file.
- The codebase attribute points to the base URL of the applet. It can be used in combination with the code attribute to point to an applet's class file(s), but does not have to be present in the HTML code.

6.2.3 AREATAG

The AREA tag specifies the geometric regions of an image map and its associated links.

It has a few attributes but only one of them is of interest, the href attribute.

- The href attribute declares the supplied URL to be the *target* of this area within a *map*.

6.2.4 BASETAG

This tag is a record of the original URL of the document. This allows a web master to move the document to a new directory or even a new site and have relative URLs access the appropriate place with respect to the original URL. If the BASE element is absent the document viewer assumes the base URL to be the one it used to access the document.

This tag has only one attribute, href.

- The href attribute points to the base URL where relative links will be appended to so a viewer may find the resources pointed to by them.

6.2.5 BLOCKQUOTE tag

This tag is used for long quotations. Many authors have used BLOCKQUOTE as a mean of indenting blocks of text. It does not have to use its only attribute, which is cite.

- The cite attribute designates a URL that points toward a resource supposed to contain an informational document about the citation.

6.2.6 BODY tag

This tag represents the body of a document. The document's content may be presented by a user agent (a browser) in a variety of ways. For example, for visual browsers, one can think of the BODY as a canvas where the content appears: text, images, graphics, etc. For audio user agents, the same content may be spoken. This tag has many attributes but only one is of interest for the author, the background attribute.

- The background attribute's value is the URL of the graphic that will be tiled as the background of the page. The user will not see this background for non-compliant browsers; if image loading is turned off; or if the user has overridden the background images in their preferences.

6.2.7 FORM tag

This tag is placed about a section of an HTML document that includes FORM elements. Other BODY tags can take place in a form, and multiple forms can occur in a document, but forms cannot be nested. There are two attributes crucial to forms, but only one is required for this software, the action attribute.

- The action attribute indicates the URL of the processing gateway. This URL will point to a program rather than a document. This program will receive the contents of the form in one of two ways depending on what value is specified for the **METHOD** attribute which is another attribute belonging to the FORM tag.

6.2.8 HEAD tag

This tag contains information about the present document, such as its title, keywords that may be useful to search engines, and other data that is not considered document content. Elements within the HEAD are usually not displayed. This tag has an attribute called profile, which is very rarely used.

- The profile attribute designates a URL that points towards one or many profiles of META information (META elements which are primarily used by search engines to index a web page).

6.2.9 IMG tag

This tag is used to insert an image into the present document. Among the many attributes, one is used, it the src attribute.

- The src attribute designates the URL pointing to an image file.

6.2.10 INPUT tag

This tag allows the easy input of a single word or line of text, and normally defaults to a width of 20 characters. It has many different attributes, but one that is rarely used is of interest, the usemap attribute.

- The usemap attribute designates the URL of the reactive image to which this element is associated.

6.2.11 LINK tag

This tag provides a media independent method for defining associations with other documents and resources. A few browsers as yet have benefited from it. It is also used to designate authorship, associated indexes, glossaries, etc. Links can also indicate the tree formation in which the document was authored by pointing, for example, to the *parent*, *next* or *previous* documents.

The attribute of interest is the href attribute.

- The href attribute specifies a URL designating the linked resource.

6.2.12 SCRIPT tag

This tag puts a script inside the document and may appear many times inside the HEAD and/or BODY of the document. The attribute of interest is the src attribute.

- The src attribute specifies the location of an external script.

CHAPTER 4

ANALYSIS

7 Problems and solutions encountered during research

7.1 HCI

During research carried out on Human Computer Interaction, it became evident that a great deal of experience is required to build a system which is easy to use, easy to learn, handles errors transparently and is efficient in doing what ever it is supposed to do. The lack of experience in designing good user interface that unites all the above qualities is not to be underestimated. There are not many solutions to this problem, but using common sense and spending the necessary time in laying out the different components needed for the application should give a good result. As mentioned on page 15, even the best user interface will not please every users, so the best possible layout has to be implemented in order to satisfy most people.

7.2 Similar products

When other products where looked at, it quickly became apparent that only a very small percentage of similar product could be examined. The research for this part of the project was done exclusively using the Internet. It was felt that it was the best way to go about it for the following reason. When a search engine on the World Wide Web returns any information, it does so by sorting it from the most relevant and up to date one to the less relevant available. This meant that the most popular or relevant software would be looked at and tested. This also meant that only a handful of these applications were needed to discover the functionality of most of them and get closure on what needed to be done. However, the information gained is not as extensive as it would be if this project were to be carried out by a team of developers and researchers. To solve this small problem, the information gained will be used as wisely as possible, and assumptions will most probably be made to ensure the smooth running of the application's design stage.

7.3 Robot Exclusion Standard

This standard, which should be used by every 'Web Robots', does not seem to pose any implementation problems. It is yet an early stage in the development of this application, but it is felt that a simple data structure accompanied by good algorithms will suffice in ensuring the good behaviour (Netiquette) of the future application over the different networks it will encounter.

7.4 HTML tags

A few of the tags found to have attributes pointing to resources seem to be very rarely used in web sites implementations. After having spent time thinking about the validity of such attributes being implemented into the future software, it was decided that for the sake of completion that all the relevant tags and their resource attributes should be implemented. The reasons are as follow. It is assumed that the amount of time required to incorporate such features in the system will be similar if there are ten of those features as if there was two or three, so no valuable time would be lost in that way. It is also felt that along the same line as the implementation, the design of possible diagrams at the design stage will be very quick once one of these features has been diagrammatically exposed.

7.5 Summary

In all, a few problems where encountered during the research part of this project, but nothing that cannot be solved easily. With a little common sense and decision-making, solutions should become clear as the software is being implemented.

8 Programming language analysis

Choosing the right language for implementing a system is crucial, this section will look at two different languages and a choice will be made depending on the results of this analysis.

8.1 Java and C

The Java language is a rather new language compared to the C language, which has been around for quite a lot of years. The next sections show a detailed overview of the two languages, their differences and their similarities, after which a decision will be made on which one to use.

8.1.1 Java

A company called Sun Microsystems® introduced the Java language on May 23rd 1995 (David Flanagan. *Java in a Nutshell 2nd Edition*. O'Reilly, 1997). The original Java version 1.0 was small compared to what it is nowadays. The Java language rules have changed since its beginning and the version out at the moment is version 1.3. Although Java has progressed a lot, it is important to bear in mind that it is just a programming language like many others, and its API is just a collection of class libraries. Java is recognised as an interpreted language, which means that a Java program can be written on any machine, and executed on any platform that has a Java Virtual Machine (JVM) installed. This is possible for the reason that the Java compiler generates *byte code*, which is free of platform and can be run on any JVM. A JVM acts as an interpreter between the Java byte code and a computer's operating system. This scenario has its advantages of being a portable, platform independent language, but it also has its disadvantages. The generated byte code means that when loading up a program written in Java, the JVM requires to be loaded up and the byte code interpreted previous to execution, which increases the running time of many Java programs (David Flanagan. *Java in a Nutshell 2nd Edition*. O'Reilly, 1997). Just In Time (JIT) compilers help speed up the running time, but then increase the program size.

There are many features in Java that will not be included in the application, such as many of the APIs (Java Applets, Java Beans etc...). However, there are some characteristics of Java that are part of the language but not evident in the syntax. One such feature is the *Garbage Collection*, in which the JVM will clean up memory of objects that are no longer used. Although a call to *System.gc()* can be made to force the garbage collector to clean up memory, it is not 100% sure that the memory will be cleaned up. Implementing this directly in C would imply creating functions that would keep a table of objects (structures in C)¹ in memory. *Threads* and *Exceptions* are also a part of Java and these features make it a really attractive language to use.

8.1.2 C

The C language has been around for many more years than Java. ANSI C was adopted in 1983 to normalise the language, making it feasible to write portable² programs. The C language is recognised and used widely along with the availability of C compilers on almost all platforms (Ellis Horowitz, Sartaj Sahni, Susan Anderson-Freed, *Fundamentals of Data Structures in C*. Computer Science Press, 1993.). The C language is a very popular programming language, and is one of the reasons why it is often used.

C is not an Object-Oriented language; Java was based on it in terms of primitive data types, control statements, operators, operator precedence etc. Compiled C programs are platform dependent, this is a disadvantage in terms of portability, but a benefit in terms of speed. Compiled C programs run much faster because they have been optimised for the defined platform on which it was compiled on and do not need an interpreter to run.

8.2 Comparison of Java and C

Java has been compared to C due to the fact that the Java language has had many of its characteristics taken from C. Such characteristics of the Java language that

¹ This sentence may confuse novices. The C language is not object-oriented like Java is.

² For a C program to be portable, it would have to be compiled on the platform onto which it is to run.

compare to C are the primitive data types, operators and operator precedence, and also most of the control statements. Java is an object-oriented language and C is not. Object-oriented programming is associated to programming in C++, a progression of C. Although Java borrows many terminology and keywords from the C++ language, one must not see Java as being the same as C++ (David Flanagan. *Java in a Nutshell 2nd Edition*. O'Reilly, 1997)

8.2.1 Primitive data types comparison

Figure 3 (table taken from David Flanagan. *Java in a Nutshell 2nd Edition*. O'Reilly, 1997) shows the comparison between the Java primitive data types and the C primitive data types. As can be seen, the *boolean* and *byte* types have been added to the Java language. Also it is important to note that the size of all Java primitive data types are known in advance and not system dependent like some C primitive data types. The *int* type in C may be 16, 32 or 64 bits depending on the machine on which it is used. Java and C primitive data types both contain unknown data when they are first created, Java does not allow for these variables to be used with out prior initialisation of the variable.

Type	Contains in Java	Size	Type	Contains in C	Size
boolean	true or false	1 bit	N/A	N/A	N/A
char	Unicode character	16 bits	char	signed character	8 bits
byte	Signed integer	8 bits	N/A	N/A	N/A
short	Signed integer	16 bits	short	signed integer	16 bits
int	Signed integer	32 bits	int	signed integer	system dependent
long	Signed integer	64 bits	long	signed integer	32 bits

float	IEEE 754 floating-point	32 bits	float	float	32 bits
double	IEEE 754 floating-point	64 bits	double	double	64 bits

Figure 3: Data types

8.2.2 Operator precedence comparison

The operators and precedence of the operators in Java and C are more or less identical (see Table 2). Java has a few operators that are not represented in C. Java doesn't support the comma operator used to join two expressions together, it also does not use the reference/dereference operators *, -> and &. Java also does not consider the. (dot) operator in C as an operator, but rather a field access. Java has added the + (string concatenation), instanceof, >>>, & and | (operators for boolean type) to it's operators which C does not have (David Flanagan. *Java in a Nutshell* 2nd Edition. O'Reilly, 1997).

Prec.	Assoc.	Operation Performed	Operator in Java	Operator in C
1	R	pre-or-post increment (unary)	++	++
	R	pre-or-post decrement	--	--
	R	(unary)	+, -	+, -
	R	unary plus, unary minus	~	~
	R	bitwise complement (unary)	!	!
	R	logical complement (unary) casting operator	(type)	(type)
2	L	multiplication, division, remainder	*, /, %	*, /, %
3	L	addition, subtraction	+, -	+, -
	L	string concatenation	+	N/A

4	L	left shift	<<	<<
	L	right shift with sign	>>	>>
	L	right shift with zero extension	>>>	N/A
5	L	less than, less than or equal	<, <=	<, <=
	L	greater than, greater than or equal	>, >=	>, >=
	L	type comparison	instanceof	
6	L	equal	==	==
	L	not equal	!=	!=
	L	equal (same object)	==	N/A
	L	not equal (different object)	!=	N/A
7	L	bitwise AND	&	&
	L	boolean AND	&	N/A
8	L	bitwise XOR	^	^
	L	boolean XOR	^	^
9	L	bitwise OR		
	L	boolean OR		
10	L	conditional AND	&&	&&
11	L	conditional OR		
12	R	conditional (ternary) operator	?:	?:
13	R	assignment	=	=
	R	assignment with operation	*=, /=, %= +=, -=, <<=, >>=, >>>=, &=,	*=, /=, %= +=, -=, <<=, >>=, N/A &=,

			$\wedge=, =$	$\wedge=, =$
--	--	--	---------------	---------------

Figure 4: Operator precedence

8.2.3 Control statements comparison

There are many control statements in Java that are identical to the ones in C, but there are also more control statements that have been added that are not represented in C. The if, else, while, do/while statements are the same in C and Java. The difference is that Java's boolean type cannot be cast to another type. Java's boolean false is not like the value 0, and boolean true is not the same as a non-zero value. The switch, break and continue statements are other statements that work very much like those in C. The for loop does differ a little, the difference is that a variable can be declared within the initialisation part of a for loop (much like in C++, but not in C) (David Flanagan. *Java in a Nutshell 2nd Edition*. O'Reilly, 1997).

Extra statements, which Java has, are the try/catch/finally statements, which are used to deal with exceptions, C does not deal with exceptions. Also, the synchronized statement is another addition to the Java language due to the fact that Java is a multithreaded system (David Flanagan. *Java in a Nutshell 2nd Edition*. O'Reilly, 1997).

8.3 Imports, includes and other differences

Instead of using *#include* like in C, Java uses the keyword *import* to carry out the duty of including packages from different directories so that the methods can be referred to with short names, rather than the long extensions required. Variable declarations in Java can take place more or less anywhere within a method body or block within a class. Forward referencing will cause problems in a generated C version if they were left where they were. The reason is that C does not allow variables to be declared anywhere in the program, all declarations must be at the beginning of a block, like at the start of a function for example. Forward referencing of a variable in Java that has not been initialised cannot be done. Method overloading is also not supported by C, which in itself presents a problem

when it comes to generating C source code (David Flanagan. *Java in a Nutshell* 2nd Edition. O'Reilly, 1997). Several mechanisms exist to allow for automatic documentation to be made of a C program, programs exist in which comments added to the source code are parsed and documentation files generated from the output. With the Java distribution comes a tool called *javadoc* that does the same thing. All the programmer needs to do is add comments in a pre-agreed manner so *javadoc* can parse the source code and generate automatically some files describing the functionality of the program. Those files are usually read in a browser. An example of such files can be found at the following location: <http://java.sun.com/j2se/1.3/docs/api/index.html>

8.4 Which language will it be?

The software could actually be written in either Java or C. Writing it in C would be, it seems, more complicated. Java has already got in its API, objects that deal directly with network connections, such as the classes contained within the *java.net* package, and also classes contained in the *javax.swing.text.html* package, which deal with parsing html documents. Such libraries do exist for the C language, but are cumbersome to use and it is felt that it would take too much time to learn how to use them.

Another important factor is that the author has had experience in programming with Java, but practically none in programming with C.

So the choice is clear, Java will be used to implement the software.

9 Functional requirements

This chapter discusses in details the future software's functional requirements.

9.1 Network connections

The first important aspect is that the application must be able to connect to the Internet. For this there are a wide variety of ways of making connections, but as the Java programming language is going to be used to build this application, there should be no problems. Java has a lot of different objects in its API to make network connections simple to use. The simplest one is a socket object, it connects to a given host on a specified port, and from there, I/O streams can be use to send and receive data.

9.2 Robot Exclusion Standard

The second most important feature is the implementation of the robot exclusion standard. Without it, the application would not be worthy of being called a robot. As explained on page 26, any respectable web robot should be able to recognised if the server it requires information from is willing to interact with it or not. Most

webmasters do not implement the standard, but the ones who do should not be ignored. A simple text parser needs to be implemented to make sure that Javasp3 will be allowed to proceed.

9.3 Data retrieval

Another important thing to look at is the way in which the application is going to retrieve the html data to be parsed. As described on page 27, Java I/O objects and methods are plentiful for this sort of work. Once the data has been retrieved, it has to be parsed to look for information of interest, the tags and their related attributes pointing to resources. Again, Java provides mechanism by which an html document may be parsed for these tags, and once found, their attributes checked to see if they comply with the requirements.

9.4 Data parsing

The goal of the software is to look for bad or broken links within the html data retrieved. For this a parser needs to be implemented, this parser has to look for predefined tags within the html and then check if they posses any link attribute pointing to resources. Once a link has been found, it then must be kept in an accessible place for other objects use.

9.5 Link history

At this moment in time, it is felt by the author that at least one object will have to be built to keep information about each connections made and each resources encountered along the way. For example, it may be interesting to keep a record of where a certain resource comes from, if it has a parent resource and if it has, which one is it, and so on. This would for example help to avoid checking for the same link twice, it would also permit for the easy pin pointing of broken links within the checked web site and allow to build an easy to understand report layout for the user.

³ Javasp3 is the name chosen for the application; it spies on web sites and will be built using the Java programming language.

9.6 Depth

After examining the different products available, it seems that most of them implement the idea of depth. Below is a small diagram explaining what is meant by depth. The diagram represents a vertical depth; there can also be horizontal depth, which is usually called breadth.

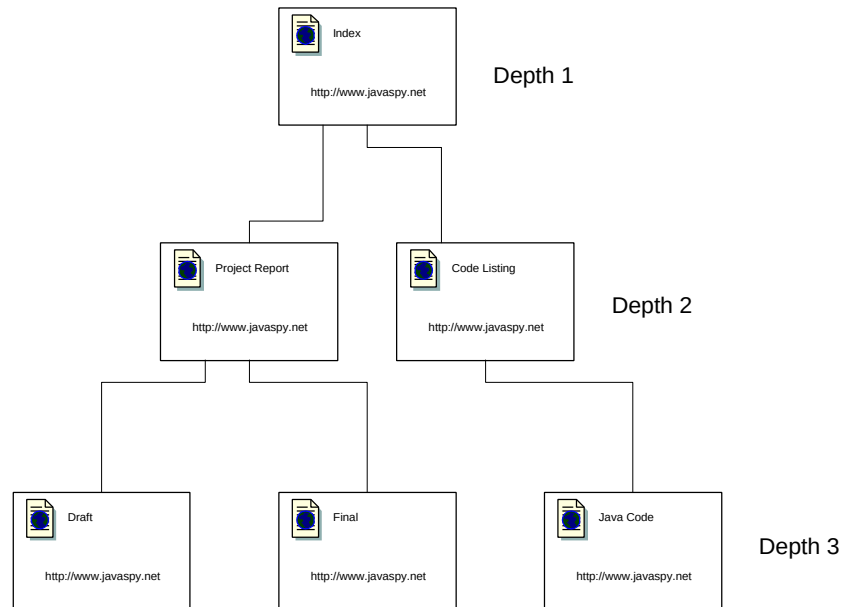


Figure 5: Depth representation

Depth functionality will have to be added to the software, to keep a certain standard of quality and usability. It would be disastrous if a robot were allowed to go through a web site without any limit as to the depth to explore. This could mean that a large site for example like www.microsoft.com which has thousands and thousands of pages, would be hit very hard indeed, and also it could also mean that a web robot would never see the end of the site and eventually give up due to lack of memory without giving any satisfactory output to the user.

9.7 Graphical user interface

One of the obvious features of this software is its graphical user interface. Java provides a wide array of classes, which make the implementation of such interfaces easier than other languages. Swing, which is one of Java's graphical components libraries, will be used to build the interface.

9.8 Reporting findings

The objective of JavaSpy is to let its user know if any links within a web site are not working anymore. For this, a simple report will be built and displayed to the user as the application runs. The report should contain substantial information such as the title and URL of the web site visited, the depth used, and obviously the URLs of the broken links found as well as the page on which they are situated. Another functionality would be to send the report to the user of JavaSpy via email, which would allow the robot to execute while the user does something else or goes away from the computer on which the program is running. This functionality may or may not be added; it will be decided at a later stage if it should be.

9.9 Other features

It may come up during the development of JavaspY that other types of functionality are needed to satisfy its implementation. If this is the case, the extra functionality will be documented during the development stage.

10 Design method

A method is needed to design and implement JavaSpy, in this section, two different methodologies will be analysed and one chosen for the work to be done.

10.1 Jackson system development

Jackson System Development (JSD) is a development method originally explained by Michael A Jackson in his book 'System Development' which was published in 1983. JSD grew out of Michael Jackson's structured program design method, JSP, and has added to, and refined, the principles of JSP.

JSD has been given a lot of attention as a real time development method because it supports the modelling of synchronized processes and their communication. It can also be used to develop object-oriented systems because it models entities and actions on those entities.

From the technical point of view there are three major stages in JSD, each divided into steps and sub-steps, a description of each can be found below.

10.1.1 The modeling stage

In the modelling stage of JSD, a description is made by the developers of the inner workings of the business, organisation or already existing system that the system will be built for. To make this description, an analysis must be made; choosing what is related to the system and drop what is not related. The organisation or system should be considered, as it will be looking like in the future and not as it looks like at present. The model description has to be written accurately. This accuracy obliges the developers to ask in depth questions to the future users of the system. This means that communication and understanding between developers, users, and any other parties involved with the new system must be very good. The model description is made of actions, entities and system-associated information. An action is an event, usually outside the system, which is relevant to the system. The first use of JSD is to make a list of actions with detailed explanation of these actions, and their related attributes. Diagrams illustrate ordered relationships

between actions. The diagrams give details about the entities, individuals or any other things that the system needs.

The result of the modelling stage is a set of tables, definitions and diagrams that describe:

- In user terms, exactly what happens in the organization and what has to be recorded about what happens.
- In implementation terms, the data structures and their contents.

10.1.2 The network stage

In the network stage a precise representation of the system's functionality is drawn, as well as the outputs that are to be created to feed the system and the way the structure will come out to the user. Developing one program for each entity that was defined during the modelling stage makes the starting point to the network. The network carries on being built up by adding new programs and connecting them up to the already implemented network. New programs are added for the following reasons:

- To collect inputs for actions, check them for errors, and pass them to the entity programs.
- To generate inputs for actions which do not correspond to external events.
- To calculate and produce outputs.

There are two different ways of linking programs together in a network. These are by data streams (represented by circles on a network diagram) and by state vector inspection (represented by diamonds on the same diagram). The entity programs are very important for the construction of the network. To describe the system, an entire set of network diagrams gets drawn. The diagrams are supported by information in the form of text, describing the contents of the data streams and state vector connections. The new programs that are added to the network are defined using the same diagrammatic notation used to describe the ordering of actions. These new programs are designed using the JSP (Jackson Structured Programming) method, which is a subset of JSD.

10.1.3 The implementation stage

The final system is the result of the implementation stage. This stage is the only one directly concerned with the machine architecture and the associated software the system is to run on. As well as producing and testing code, the implementation stage covers physical design issues. In particular it covers:

- Physical data design.
- Reconfiguring the network by combining programs.

Physical data design is about the design of files or databases. The details of database design depend on the database management system being used. However, the necessary information about the application is available from the network stage. The result of the network stage is a highly distributed network of programs. Programs get converted into subroutines very often, as it is more convenient and efficient, this in turn has the effect of combining several programs into one, so that a portion of the network is implemented as a program on its own.

10.2 The UML

The UML (Unified Modelling Language) as its name suggests, is more of a language than a methodology. It is used for detailing, picturing, building, and documenting the objects of an application or system's processes. Rational Software Corporation and three of the most famous software development methodologists conceived it; they are Grady Booch, James Rumbaugh, and Ivar Jacobson (the Three Amigos). The UML is relevant to many different forms of system development. It is nowadays very often used for developing Object-Oriented systems; some of the most important software companies and organisations use it regularly. The UML is made of graphical elements, which joined together form diagrams, and because it is a language, the UML possesses rules for joining these elements together. The function of diagrams is to show the system in different views, and a collection of these views is referred to as a model. On the next pages are descriptions of the most often used diagrams in the UML and what they stand for.

10.2.1 Class diagram

A class is a type or collection of items, which have common actions and have related attributes. An example would be that of a bird class, everything in the bird class has attributes such as species, wingspan, age span, colour, sex, etc.... Actions for attributes in this class comprise of the following functionalities: set colour, get colour, set sex, get sex, set species, get species and so on. Figure 6 shows how the UML notation looks like when it captures these actions and attributes. A class is represented by a rectangle split into three regions. The first region holds the name of the class; the second region holds the attributes, and the third the actions. A class diagram is made of two or more of these icons joined by lines that illustrate how the different classes interact with each other. A class diagram grants software developers illustrations from which they can work. It also allows software analysts to communicate easily with their clients.

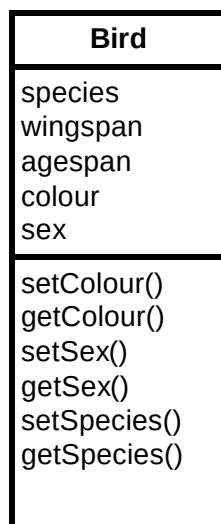


Figure 6: Class icon

10.2.2 Object diagram

An object is an instance of a class; it contains values for the attributes and functionality for the actions.

For example, a bird may be from the robin species, be a female, and have an age span of 7 years. Figure 7 shows how the UML notation looks like when it represents an object. The name of the class is on the right hand side of a colon and

the name of the instance is on the left hand side. The two combined together make the object name and is underlined.

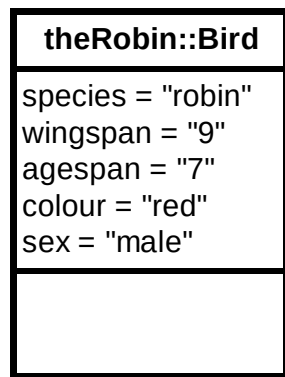


Figure 7: Object icon

10.2.3 State diagram

An object is in a particular state at any given time, for example a bird could be walking, flying, eating, sleeping, etc.... Figure 8 shows how the UML notation looks like when it shows the state of the bird object transitions from one action to the other. The state diagram has a figure at its top, it represents the state at which the object starts and a figure at its bottom showing its last state.

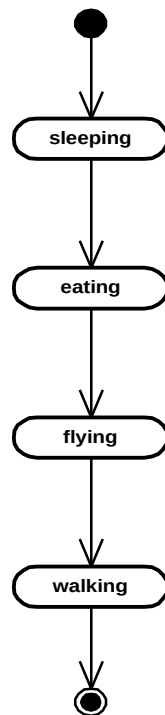


Figure 8: A state diagram

10.2.4 Use Case diagram

A use case diagram shows a system's action from a user's perspective. This sort of diagram is very important to a software developer; it helps to capture the future system's requirements from a user's viewpoint. It is a very important diagram when building an application that non computer literate people will use. Figure 9 shows how the UML notation looks like when a user is interacting with the program. The person icon represents the user, but it can also represent another part of the system. The ellipse represents the use case.

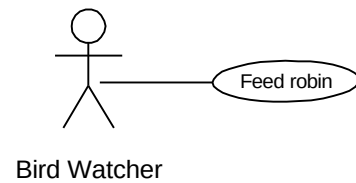


Figure 9: A use case diagram

10.2.5 Sequence diagram

The sequence diagram represents the time-based dynamics of the interaction between different objects within a program. Carrying on with the bird example, the components of the bird include a digestive system, a vocal system, a vision system etc; these are also objects in their own rights. What would happen when the vision system is invoked? A sequence of steps would go as follow:

- Light enters the retina.
- A nerve transmits data to the brain.
- The brain processes the data.
- The data goes to another part of the brain to be used accordingly.
- Light entry restarts after an eye blink.
- The nerve transmits data.
- The brain processes it and sends it to another part of the brain.
- The bird goes to sleep.

- The eye shuts.

Figure 10 shows how the UML notation looks like when a sequence diagram shows the interaction between the retina, the nerve and the brain. The different entities are represented at the top of the diagram by rectangles. Time progresses from top to bottom.

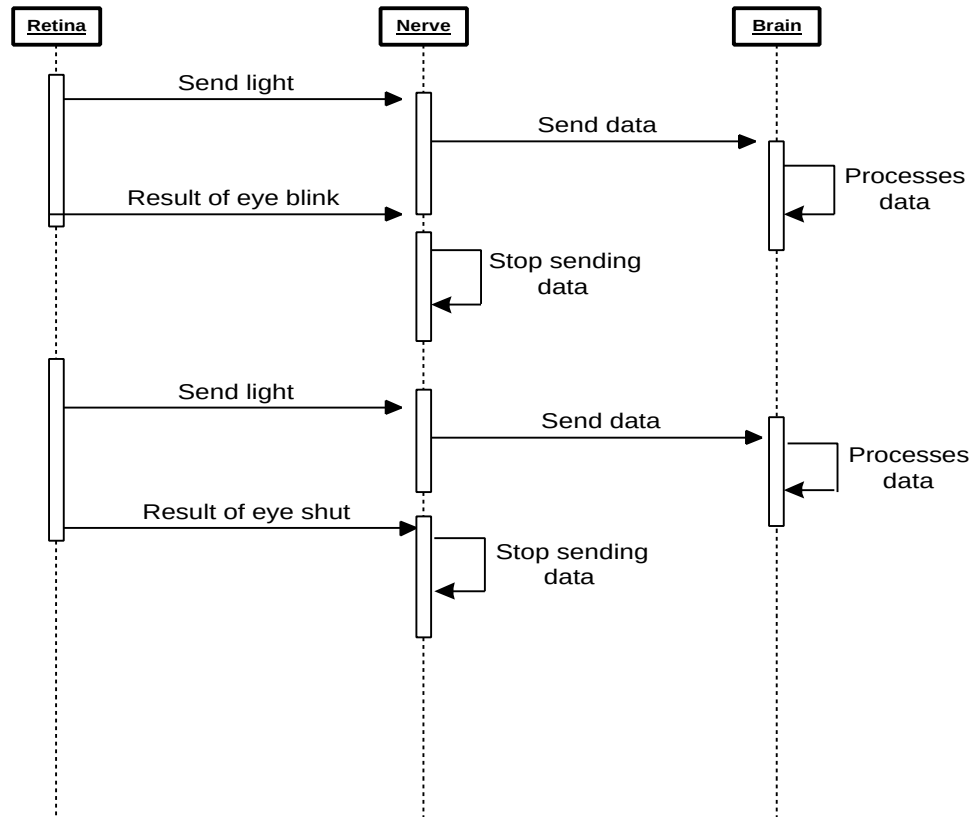


Figure 10: A sequence diagram

10.2.6 Activity diagram

The activity diagram represents the activities that take place in sequence within a use case or an object's actions. Figure 11 shows how the UML notation looks like when representing this sequence.

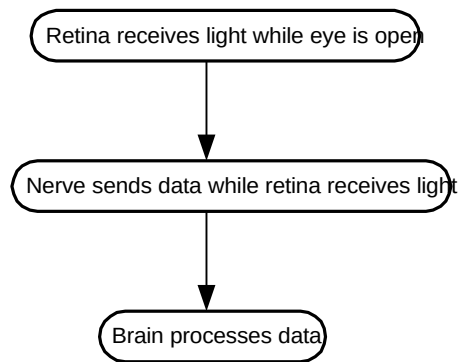


Figure 11: An activity diagram

10.2.7 Collaboration diagram

To achieve a system's goal, its building blocks work jointly, and the UML has a technique to represent this. Figure 7 shows how the UML notation looks like when representing this. An additional timer object has been added, after a while, the timer stops the eating process and starts the flying one.

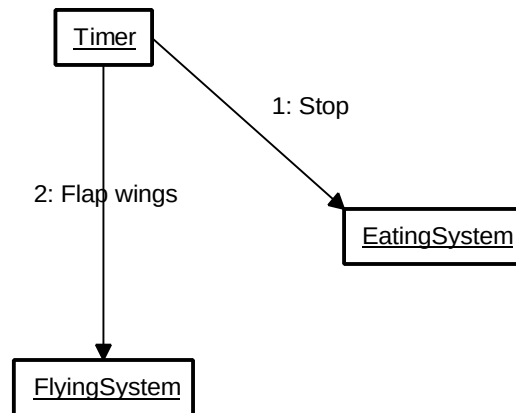


Figure 12: A collaboration diagram

10.2.8 Summary

There are many other elements and diagrams in the UML notation, but the only the ones that seem of interest to this application have been shown. It is important to be able to describe and examine a system in different views, as usually a future system has different people interested in it, some computer literate, some not. It is also important that the notation be easy to understand, as there could be

possibilities for errors in the development process if it were hard to understand how a system should act, the UML gives this easy to understand notation.

10.3 JSD or the UML?

The system to be developed is not a large system, not even a medium system. Only the author will be developing it on his own, not a team, therefore it seems that a methodology would do the opposite of helping in the growth of the system and impair its development in terms of time, complexity, and paper work. JSD will be left aside for this exercise and the UML will be used to explain, using different views, how the system and its components will be working and interacting with each other and the outside world. As mentioned earlier, the UML has more components and diagrams available to its notation, if the need is felt that some of these elements need to be used during the development stage, then a short description will be made of them as was done previously with other of its notation elements. A use case diagram will be used to show the interaction between the user and the program. Class diagrams will be drawn for each class present in the software. Sequence and collaboration diagrams will also be drawn to show the internal functionality of the produced software. Alongside the UML, a prototyping approach will be used. This decision was taken because this is the first time that I had to produce a system from start to finish, and errors will most probably occur frequently, therefore when code is produced, it will be tested straight away and if any changes need to be made, they will be made immediately to avoid having a lengthy debugging session.

CHAPTER 5

DESIGN

&

IMPLEMENTATION

11 Testing and evaluation

This section is only a brief overview of the methods that will be used to test JavaspY and evaluate it.

11.1 Testing

Testing of the source code will be done as and when required, which will mostly be as soon as the code will be written. This will ensure that errors in the program will be found as soon as possible and rectified straight away. It will also insure that no time will be lost in lengthy debugging of the final software. There are often bugs in a substantially long program, but this technique seems to be the best approach to minimising them. Once the system is finished, tests will be carried out, these tests can be found on page 89.

11.2 Evaluation

Evaluation of JavaspY will be done using two different techniques. The first one, which seems to be the better one, will be to use a comparable software and send it to find broken links in a given web site, then to send JavaspY to do the same work and compare the results from both applications. This evaluation technique will be used towards the end of the software's implementation, so changes in the code will be made if any large discrepancies occur between the two applications. The second evaluation technique will be applied once the software has been finalised. It will be given to a few people to try out with an accompanying questionnaire relating to the application's use. After receiving the questionnaires results, an evaluation of JavaspY and its usefulness in the real world will be made.

12 Hardware

An Intel machine will be used to program and test JavaspY.

The computer used is the author's own machine, and it consist of the following:

- An Intel Celeron processor running at 400 MGhz

- 256 Megabytes of RAM
- 19 inch colour monitor
- 16 Megabytes SVGA graphics card
- 10 Gigabytes Hard disk drive

There are obviously other technical specifications to be added, but the ones above are felt to be the most important ones.

The operating system used to interface with this hardware is Microsoft Windows 98.

As JavaspY will be developed in Java, it is important to know if it will run as well under a Windows OS as under a UNIX system such as Linux, or under a MAC OS. These tests will be carried out once the system is finished.

If any changes to the hardware take place during the software implementation, a note or notes will be made at the time and the reader will be informed of such changes. It is very likely that extra RAM will be added to smooth the possible use of memory hungry programming IDEs running under the Java Virtual Machine.

13 Design and implementation

In order to decide what development software I would use to design and implement the software, it is essential to look at what is available.

13.1 Tools

13.1.1 JDK 1.3 and notepad



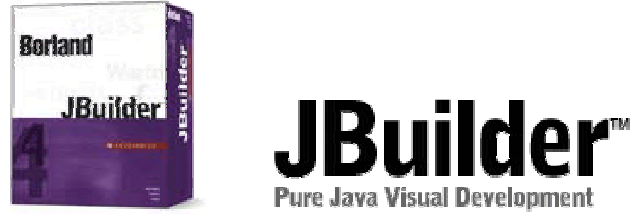
The first thing that comes into mind is to use a simple text editor such as notepad, and the JDK to write the software. The problem with this kind of approach is that the amount of code to be produced is rather large and notepad does not offer any line numbering or syntax highlighting that other editor provide.

13.1.2 JDK 1.3 and ultra-edit



Ultra edit is a very good editor, it provides what notepad does not, and even allows for commands to be added to its menu, so compiling source code and starting the program can be integrated, which makes a programmer's work a little easier. Obviously, Ultra Edit has not been written for programming in java, after all it is only a text editor, so one thing missing is syntax completion. It is important for a programmer to have access to syntax completion, because a program's source code can be very large, and remembering every attributes and methods of a class is sometimes difficult. A good reason to have Ultra Edit is that it is excellent to test little pieces of code that can later be integrated into the main program.

13.1.3 Jbuilder 4 foundation



Jbuilder 4 Foundation is a free development software from Borland. It is built in Java and is for developing software in Java. Unfortunately, Borland has added its own API and it is sometimes difficult to distinguish which API is being used. As I want to use only the sun's JDK, Jbuilder is unfortunately not for me. One good aspect of Jbuilder is how easy it is to build graphical user interfaces, and this is very important to the software being developed.

13.1.4 Together control center



Together CC is also built in Java for Java developers. It is not only a programming tool, but also entire developing software that support the UML. It allows the developer to build his/her application using the UML and the diagrams it provides. The university has an academic license for Together CC, and this makes it very attractive indeed for the work to be done. It uses sun's JDK, has syntax highlighting, syntax completion etc...

If changes are made to a diagram, the source code is automatically changed, and vice versa. Obviously, the source code generated is only a skeleton, with class names, and method name as well as variable declaration. Navigation through a project's files is made very easy. A debugger is provided which speeds up the development time and help in reducing any bugs, but unfortunately does not get read of all of them.

13.1.5 Forte for Java community edition



This software is freeware, it seems as good as Together CC, but does not support the UML, therefore does not support any diagram making. Another problem is that it seems to need a lot of resources to run, and is slow compiling and running. Overall it is a good development tool for the less fortunate programmer.

13.1.6 Summary

For developing JavaSpy, I will be using Together CC for making the diagrams and writing the source code. I will be using Jbuilder 4 for building the graphical user interface, and copy the generated code into Together CC for alteration, tuning and adding event handling methods. To test the program at any stage of the development, I will have a DOS box open, ready to call the *java.exe* command. The DOS box will be used as it was reported that running a program under Together CC may be a problem when developing software with user interfaces, a bug exists which makes the rendering engine not as good as it should be and it may mean that certain components may not be drawn correctly or not at all.

13.2 Diagrams

13.2.1 Use case diagram

To capture the high level user-functional requirements of a system, a Use Case diagram is needed. Another use that is made of the Use Case is to define the fundamental structure of the application.

The following Use Case explains what interaction a user may have with the system.

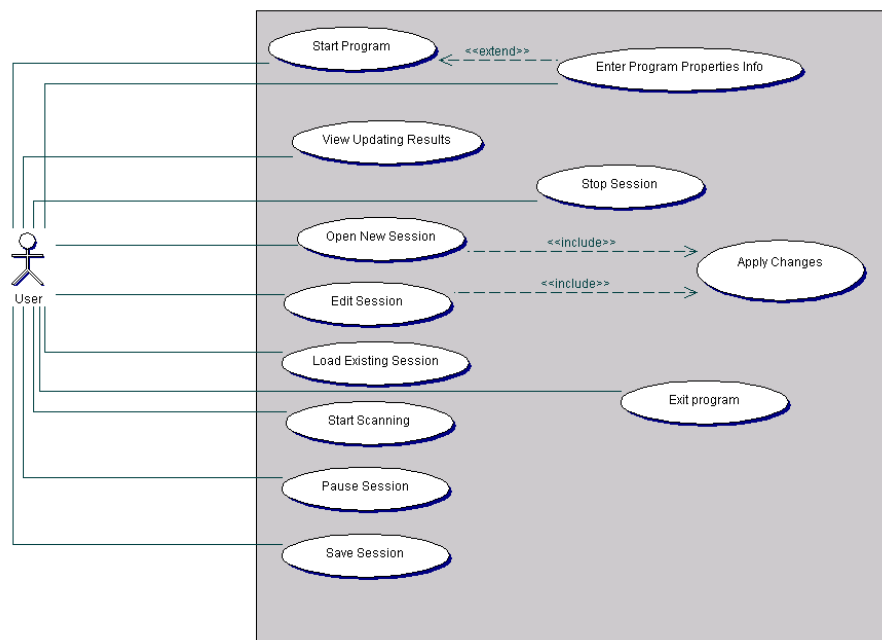


Figure 13: Use Case diagram for JavaSpy

From this diagram, you can see what a user may accomplish with the system and what interaction is needed. This first step is very important, as it will mould the following processes of development. If the interaction between a user and a system is well represented with a Use Case, it makes it much easier during the class design process to see what objects are needed. It shows what the system will do at a high level.

13.2.2 Class design

Once the Use case had been made, a set of objects had to be built in order for them to work together to form the final product. These different objects can be shown using the UML's class diagram notation. During the design and implementation of software, discrepancies may occur between the class diagrams and the programming approach used to implement the designs. This will most probably happen for this system's development, therefore only the final version of the any diagram will be included in this report.

One thing is certain, for most classes, small changes have been made during the implementation, such as removing and/or adding variables, removing and/or

adding methods. Any comments made about a class diagram, will be present below or next to it.

When thinking about what object would be needed, it occurred that the following are very important.

- Link: an object representing a link to a resource on the World Wide Web.
- Site: an object representing a web site containing links
- Scanner: an object to scan a page for links
- JavaspYGUI: an interface object, which allows a user to interact with the application.
- ProgramProperties: an object representing a data structure for safe keeping of the application's running properties.
- JavaspYMain: the active class containing the main method, which makes those objects, work together.

A class diagram represents each of these, they can be found on the next pages. Each fields and methods present in the class diagrams will be commented in the source code. The source code can be found in appendix B.



Figure 14: Link class

This class defines a link to a resource on the World Wide Web.

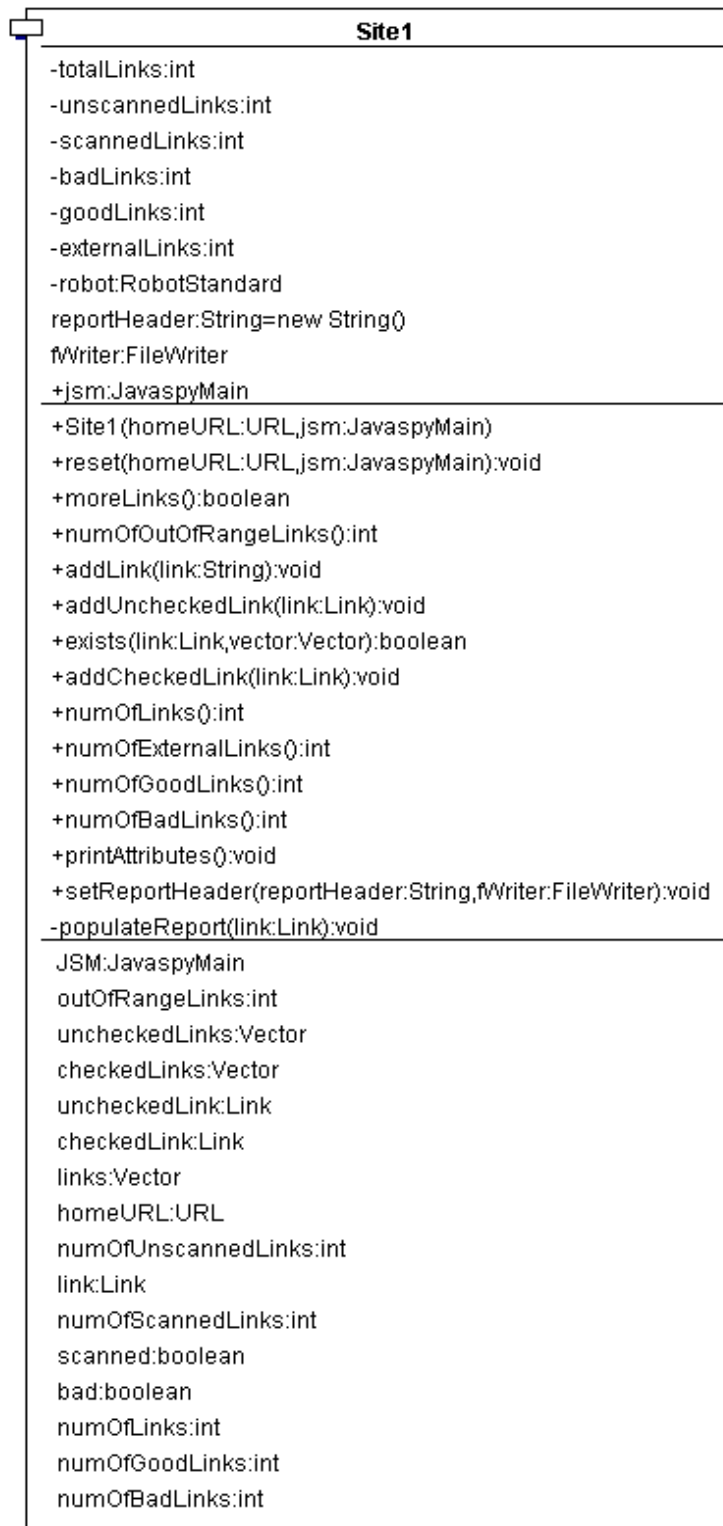


Figure 15: Site class

This object represents a web site containing Link object.

Scanner1	HTML
-jsm:JavaspyMain -parent:Link -tempLink:Link -links:Vector originalHTML:String quoteStart:int quoteEnd:int	
+Scanner1(jsm:JavaspyMain) +checkPage(page:Link):Link -retrieveLinks(page:Link):void -getPageContent(address:URL):String -getAttributes(html:String,start:int,attributes:Vector,tag:String,att:String):in -findValues(tag:String,attribute:String,attributes:Vector,s:String):void -findValuesQuote(tag:String,attribute:String,attributes:Vector,s:String):voi -findValuesSingleQuote(tag:String,attribute:String,attributes:Vector,s:Stri -isQuoteStart(attribute:String,quoteStart:int,s:String):boolean -isQuoteEnd(quoteStart:int,quoteEnd:int,s:String):boolean -findQuotePosition(quoteStart:int,quoteEnd:int,attribute:String,s:String):b -findSingleQuotePosition(quoteStart:int,quoteEnd:int,attribute:String,s:St -containsQuotes(s:String):boolean -containsSingleQuotes(s:String):boolean -findStart(start:String,index:int,html:String):int -findEnd(end:String,index:int,html:String):int -initLine(s:String):String -buildLink(parent:Link,att:String,atts:Vector):Link -isAnchor(link:Link):boolean	

Figure 16: Scanner class

This object has all the functionality for scanning a web page and extracting the links within this page.

JavaspYGUI1	JFrame
<pre> -filter:SpyFilter +fc:SpyFileChooser +lst:AdjustmentListener +jsm:JavaspYMain +statusBar:StatusBar +eb:EtchedBorder +linkPropTB:TitledBorder +scanStatusTB:TitledBorder +menuBar:JMenuBar +fileMenu:JMenu +newSessionMenuItem:JMenuItem +editSessionMenuItem:JMenuItem +loadSessionMenuItem:JMenuItem +saveSessionMenuItem:JMenuItem +exitMenuItem:JMenuItem +helpMenu:JMenu +contentsMenuItem:JMenuItem +aboutMenuItem:JMenuItem +topJPanel:JPanel +URLJLabel:JLabel +URLJTextField:JTextField +goJButton:JButton +stopJButton:JButton +splitPane:JSplitPane +listScrollPane:JScrollPane +list:JList +listModel:DefaultListModel +linkListener:LinkListListener +leftPanel:JPanel +leftTopPanel:JPanel +totalLabel:JLabel +totalResultLabel:JLabel +linksToScanLabel:JLabel +linksToScanResultLabel:JLabel +linksScannedLabel:JLabel +linksScannedResultLabel:JLabel +goodLinksLabel:JLabel +goodLinksResultLabel:JLabel +badLinksLabel:JLabel +badLinksResultLabel:JLabel +externalLinksLabel:JLabel +externalLinksResultLabel:JLabel +outOfRangeLabel:JLabel +outOfRangeResultLabel:JLabel +jScrollPane1:JScrollPane +textScrollPane:JTextArea +site:Site -screen:Dimension +cellRenderer:LinkCellRenderer +JavaspYGUI1(jsm:JavaspYMain) -initComponents():void +updateValues():void -newSessionMenuItemSelected():void -editSessionMenuItemSelected():void -loadSessionMenuItemSelected():void -saveSessionMenuItemSelected():void -stopJButtonClicked(evt:ActionEvent):void -goJButtonClicked(evt:ActionEvent):void -updateFile(file:File):File -exitMenuItemActionPerformed(evt:ActionEvent):void -resizing():void -exitForm(evt:WindowEvent):void +reset():void </pre>	

Figure 17: JavaspYGUI class

This class represents the main interface between the user and the application.

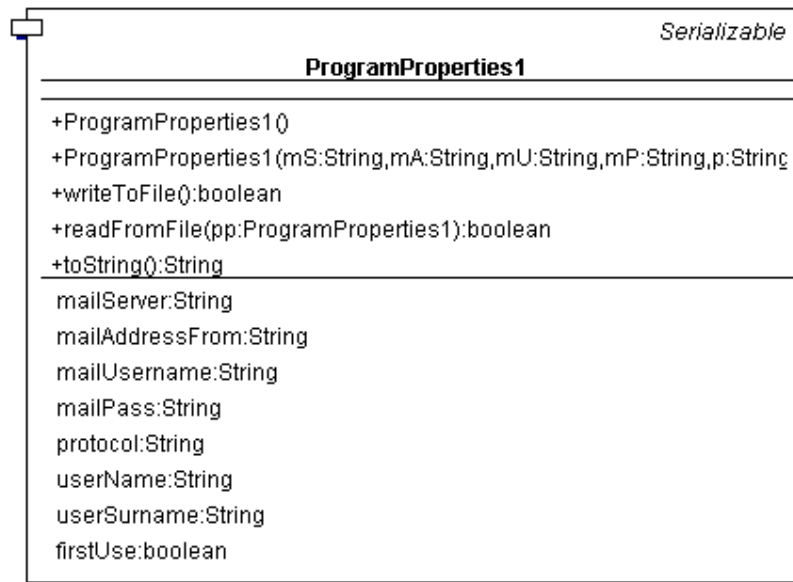


Figure 18: ProgramProperties class

A serialisable object for keeping the program's properties



Figure 19: Main application class (active class)

Once these objects were built, functionality started being added to them, but it became apparent that more objects would have to be built in order to maintain the code and make it more manageable. Adding more classes also helped in keeping a good Object Oriented concept for the whole program.

The classes added can be found below.

interface HTML 1
<u>+S ST:String=new String("<")</u> <u>+S BT:String=new String(">")</u> <u>+S WS:String=new String(" ")</u> <u>+S QUOTE:String=new String("\"")</u> <u>+S EQUAL:String=new String("=")</u> <u>+LINESEP:String=System.getProperty("line.separator")</u> <u>+PATHSEP:String=System.getProperty("path.separator")</u> <u>+FILESEP:String=System.getProperty("file.separator")</u> <u>+OSNAME:String=System.getProperty("os.name")</u> <u>+USERHOME:String=System.getProperty("user.home")</u> <u>+USERDIR:String=System.getProperty("user.dir")</u> <u>+PROXYERROR:int=1</u> <u>+CONNERROR:int=2</u> <u>+tagAtts:String[][]={{"BODY","BACKGROUND"},</u>

Figure 20: HTML interface

RobotStandard1
disallowedURL:Vector <u>robotFile:String="/robots.txt"</u> #RobotStandard1 () +reset():void +populate(homeURL:URL):Vector -buildDisallowedURL(v:Vector,homeAddress:URL):Vector +isOK(link:String):boolean

Figure 21: RobotStandard class

This class will be used to check every connection made to a URL and see if it is allowed to go on or not, therefore respecting the Robot Exclusion Standard that certain site implement.



Figure 22: JasSession class

It became apparent that the ProgramProperties class was not sufficient in keeping a record of what the program had to do and how, so this class was added. It keeps the settings for a particular scanning session. The user may change some or all the settings to make a personalised session. Default settings in a session are present too.

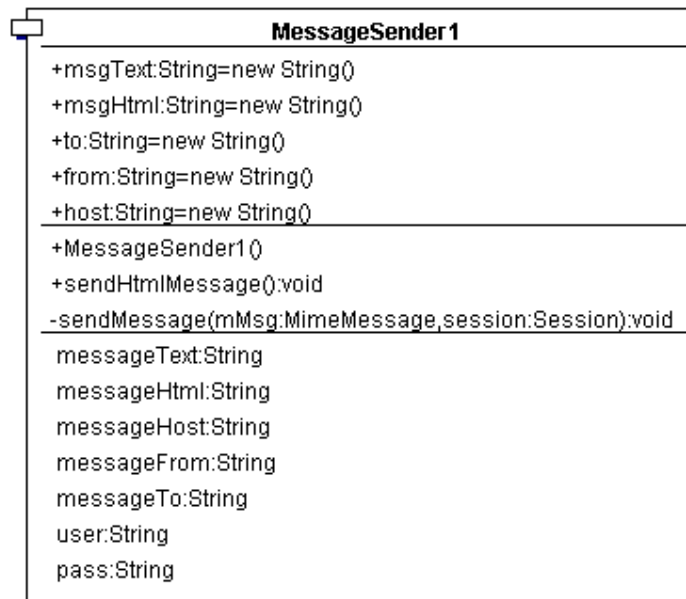


Figure 23: MessageSender class

During the analysis period, it was not sure if JavaSpy would implement sending results by email, it was decided after careful analysis to implement this class that resulted from this decision.



Figure 24: LinkListListener class

This class takes care of mouse clicks in a list of objects. If an object is clicked, the value of the object is returned.

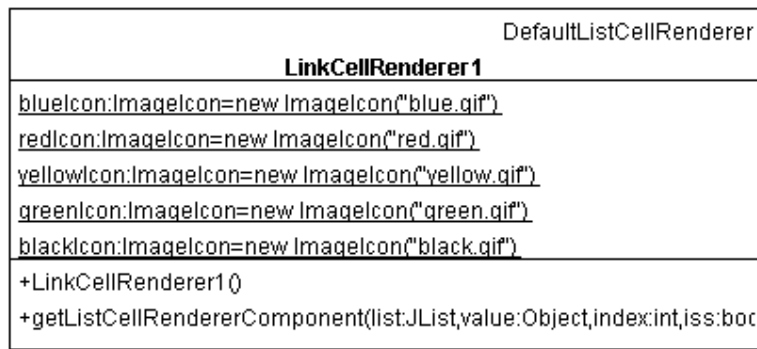


Figure 25: LinkCellRenderer class

This class takes care of displaying rows in a list object. This particular one adds a small icon in front of a line of text.

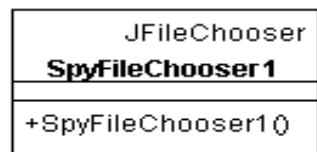


Figure 26: SpyFileChooser class

This class is used for loading or saving a file.

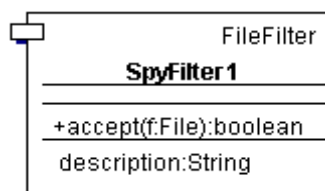


Figure 27: SpyFilter class

Works with the file chooser to allow only certain type(s) of files to be loaded or saved.

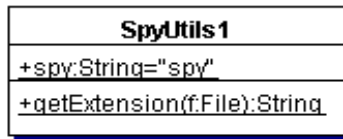


Figure 28: SpyUtils class

Works with the SpyFilter class. This class contains one string at present representing the allowed file extension. It is in this class that other extension names would be added to allow more file extension to be loaded or saved.

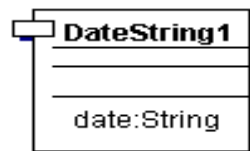


Figure 29: DateString class

A class returning a string that represents a date in a pre-defined format.

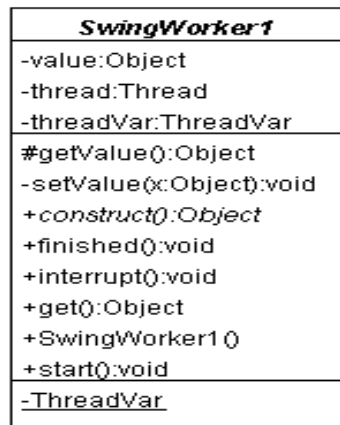


Figure 30: SwingWorker class

This class is the only one that was not created by the author. It is a utility class built by Sun Microsystems® to make threading in a graphical environment easier to work with. Information about this class may be found at <http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html>

PropertiesPanel1	JPanel
<pre> +proxyServerTitledBorder:TitledBorder +proxyAuthenticationTitledBorder:TitledBorder +userInfoTitledBorder:TitledBorder +serverInformationTitledBorder:TitledBorder +mailSettingsTitledBorder:TitledBorder +siteInformationTitledBorder:TitledBorder +reportHeaderTitledBorder:TitledBorder +proxyAddressJTextField:JTextField +proxyPortJTextField:JTextField +proxyUserJTextField:JTextField +proxyPassJPasswordField:JPasswordField +emailNameJTextField:JTextField +emailSurnameJTextField:JTextField +emailAddressJTextField:JTextField +emailServerJTextField:JTextField +emailUsernameJTextField:JTextField +emailPassJPasswordField:JPasswordField +sessionReportToJTextField:JTextField +sessionReportAddressJTextField:JTextField +sessionReportNameJTextField:JTextField +sessionURLJTextField:JTextField +sessionMaxLinksJTextField:JTextField +sessionReportHeaderJTextArea:JTextArea +sessionMaxDepthJSlider:JSlider +sessionUseMailReportJCheckBox:JCheckBox +sessionUseRobotJCheckBox:JCheckBox +sessionUseReportHeaderJCheckBox:JCheckBox +useProxyJCheckBox:JCheckBox +useAuthenticationJCheckBox:JCheckBox +jsm:JavaspMain +ProgramPropertiesJTabbedPane:JTabbedPane=new JTabbedPane() +emailJPanel:JPanel=new JPanel() +sessionJPanel:JPanel=new JPanel() +proxyJPanel:JPanel=new JPanel() +applyAndCloseJButton:JButton=new JButton() +cancelJButton:JButton=new JButton() +proxyServerJPanel:JPanel=new JPanel() +proxyAddressJLabel:JLabel=new JLabel() +proxyPortJLabel:JLabel=new JLabel() +proxyAuthenticationJPanel:JPanel=new JPanel() +proxyUserJLabel:JLabel=new JLabel() +proxyPassJLabel:JLabel=new JLabel() +emailUserInfoJPanel:JPanel=new JPanel() +emailNameJLabel:JLabel=new JLabel() +emailSurnameJLabel:JLabel=new JLabel() +emailAddressJLabel:JLabel=new JLabel() +emailServerInfoJPanel:JPanel=new JPanel() +emailProtocolJLabel:JLabel=new JLabel() +emailServerJLabel:JLabel=new JLabel() +emailUsernameJLabel:JLabel=new JLabel() +emailPasswordJLabel:JLabel=new JLabel() +emailPop3JRadioButton:JRadioButton=new JRadioButton() +emailImapJRadioButton:JRadioButton=new JRadioButton() +emailProtocolButtonGroup:ButtonGroup=new ButtonGroup() +sessionMailSettingsJPanel:JPanel=new JPanel() +sessionReportToJLabel:JLabel=new JLabel() +sessionReportAddressJLabel:JLabel=new JLabel() +sessionSiteInfoJPanel:JPanel=new JPanel() +sessionReportNameJLabel:JLabel=new JLabel() +sessionURLJLabel:JLabel=new JLabel() +sessionMaxDepthJLabel:JLabel=new JLabel() +sessionMaxLinksJLabel:JLabel=new JLabel() +sessionReportHeaderJPanel:JPanel=new JPanel() +sessionReportHeaderJScrollPane:JScrollPane=new JScrollPane() gridBagLayout1:GridBagLayout=new GridBagLayout() gridBagLayout2:GridBagLayout=new GridBagLayout() gridBagLayout3:GridBagLayout=new GridBagLayout() gridBagLayout4:GridBagLayout=new GridBagLayout() gridBagLayout5:GridBagLayout=new GridBagLayout() gridBagLayout6:GridBagLayout=new GridBagLayout() gridBagLayout7:GridBagLayout=new GridBagLayout() gridBagLayout8:GridBagLayout=new GridBagLayout() gridBagLayout9:GridBagLayout=new GridBagLayout() gridBagLayout10:GridBagLayout=new GridBagLayout() gridBagLayout11:GridBagLayout=new GridBagLayout() -parent:ProgPropDialog +PropertiesPanel1(jsm:JavaspMain,parent:JDialog) initComponents(jsm:JavaspMain,child:JPanel):void -applyAndCloseJButtonClicked():void -useProxyJCheckBoxChanged():void -useAuthenticationJCheckBoxChanged():void -useMailReportJCheckBoxChanged():void -sessionUseReportHeaderJCheckBoxChanged():void -charTyped():void -isBlank(s:String):boolean -setSessionInfo():void -setProgramPropertiesInfo():void index:int </pre>	

This class is part of the interface and displays the program's properties as well as the current scanning session properties. It was built using the Swing API from Sun Microsystems®.

Figure 31: PropertiesPanel class

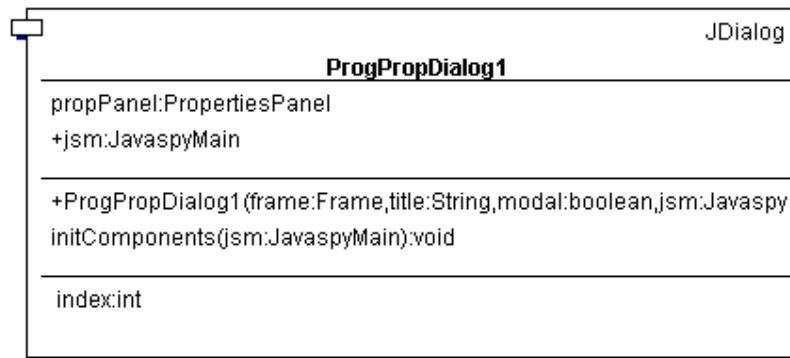


Figure 32: ProgPropDialog class

This class is used in conjunction with the `PropertiesPanel` class. It is a container frame used to display the program's properties.



Figure 33: StatusBar class

This class is also part of the interface. It represents a status bar at the bottom of the main interface. It gives information on the status of the program, and on the main settings of a scanning session.

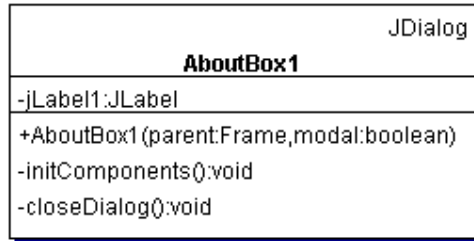


Figure 34: AboutBox class

A simple dialog box about the program itself.

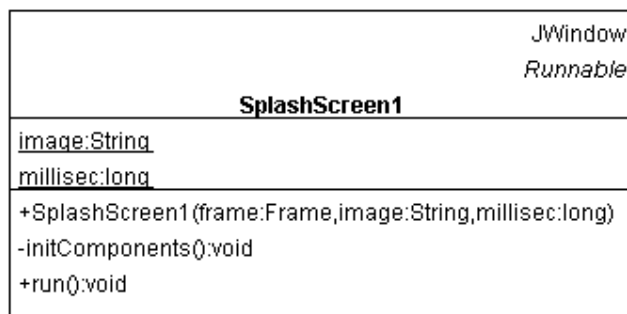


Figure 35: SplashScreen class

This class displays a splash screen when JavaSpy starts; it can be used with any .gif or .jpeg file and will accommodate any size picture.

After completion of these classes, work started on the Scanner, Link, Site and RobotStandard classes. Although the UML was used as much as possible to design this software, lack of experience building an entire system meant that I could not exactly match the design part of the system with its implementation. From time to time I had to change certain classes to accommodate for unforeseen problems occurring. Therefore, a prototyping approach was used in conjunction with the UML notation to alleviate this problem. For example, when the Scanner class was implemented, it took a few trial and error tests to finally arrive at what it is now. The same is true of the main interface. Every effort was put in this design and implementation to keep it as close as possible to the diagrams notation.

13.2.3 Sequence diagrams

Knowing what interaction happens between different objects is extremely important. Sequence diagrams are used to capture the detailed dynamic behaviour of the system. These diagrams are potentially the most complex notations the UML has to offer. They are used to decide and model how a system will do what is described in the Use Case model.

While asking lecturers and tutors on how I should go about using sequence diagram, a simple recipe started forming, and I used these different steps to create those diagrams.

- Take the Use Case description and turn it into simple pseudo code.
- Guess which classes you think might be involved - based on the content of the Use Case description.
- For each of the steps in the pseudo code, decide which of the classes should have the responsibility for doing that task.
- For each of those tasks you may want to go back and decide to break them down into a number of simpler tasks.

I used this technique to build the sequence diagrams. They can be found on the next pages. Another diagram can be used to represents interaction between objects within a system, the collaboration diagram. Collaboration diagrams are I think, easier to understand than sequence diagrams, but do not seems to show as much details as sequence diagrams, so they were not included them in this report.

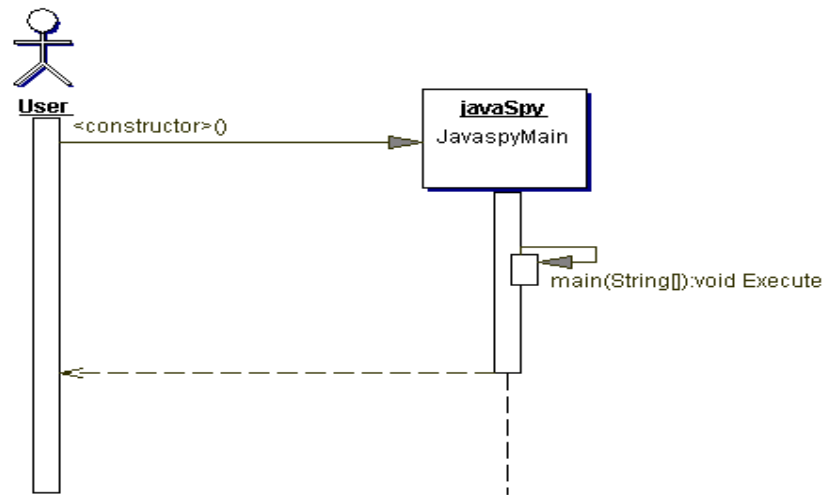


Figure 36: Start program sequence diagram

The simple task of starting a program can be easily forgotten, and look trivial to anyone, but as the Use Case models the starting of the program, I included this sequence diagram.

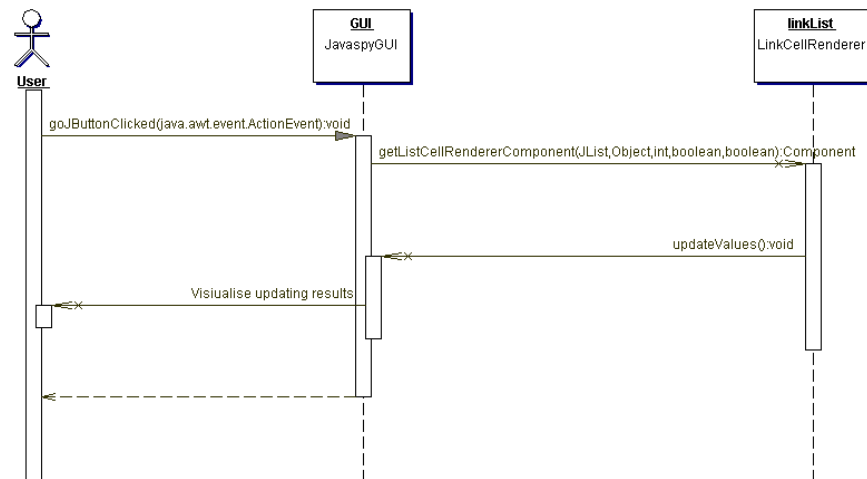


Figure 37: View updating results sequence diagram

Using the system means that results will be displayed on the screen for the user to see and act upon, this sequence diagram captures the necessary steps in starting a scan and viewing the resulting information.

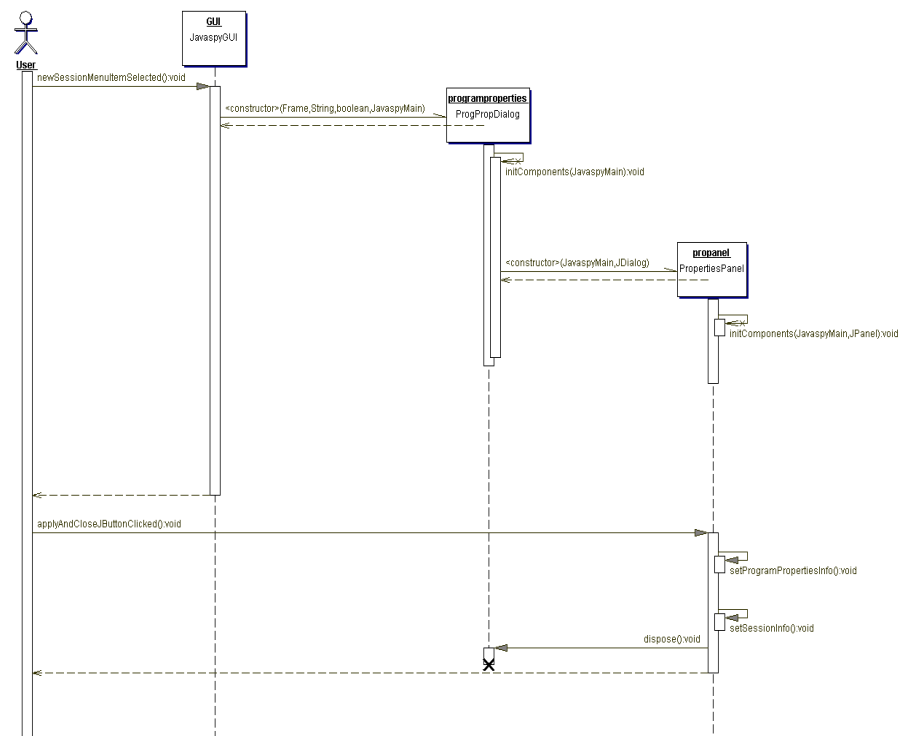


Figure 38: Open new session sequence diagram

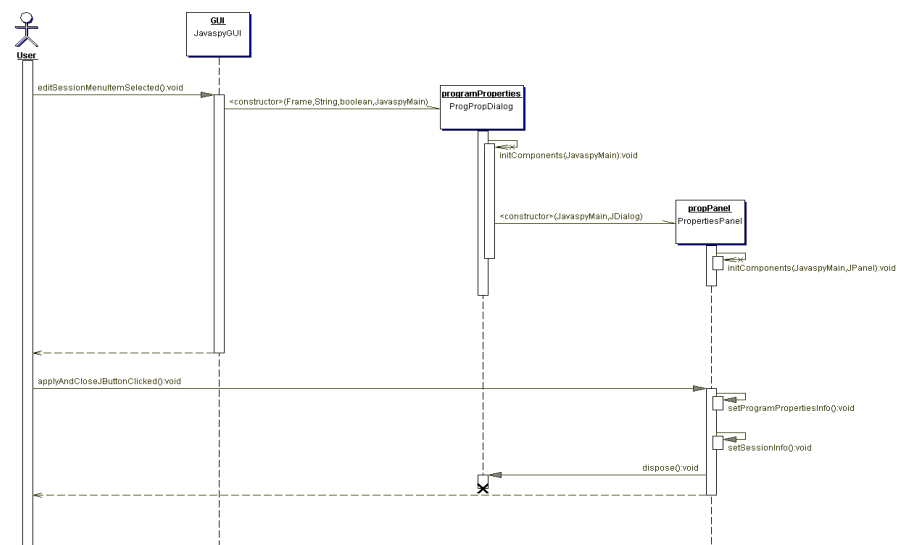


Figure 39: Edit session sequence diagram

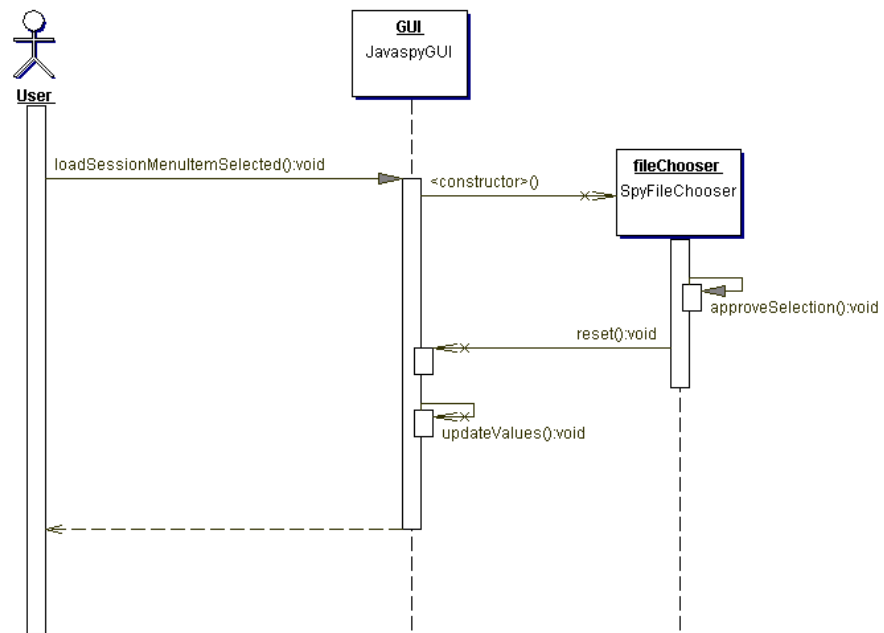


Figure 40: Load existing session sequence diagram

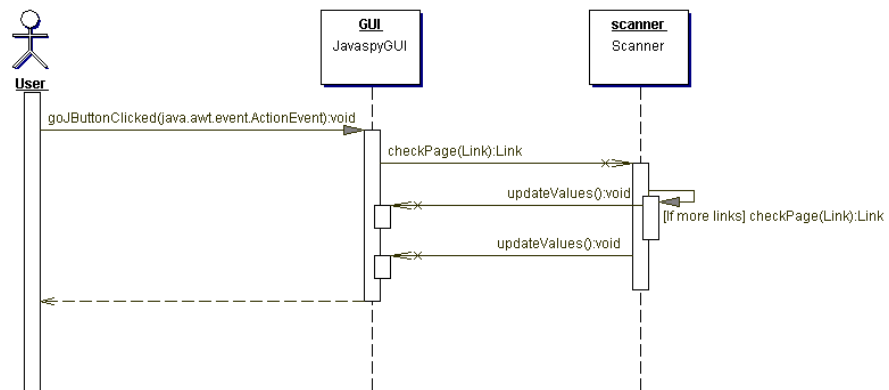


Figure 41: Start scanning sequence diagram

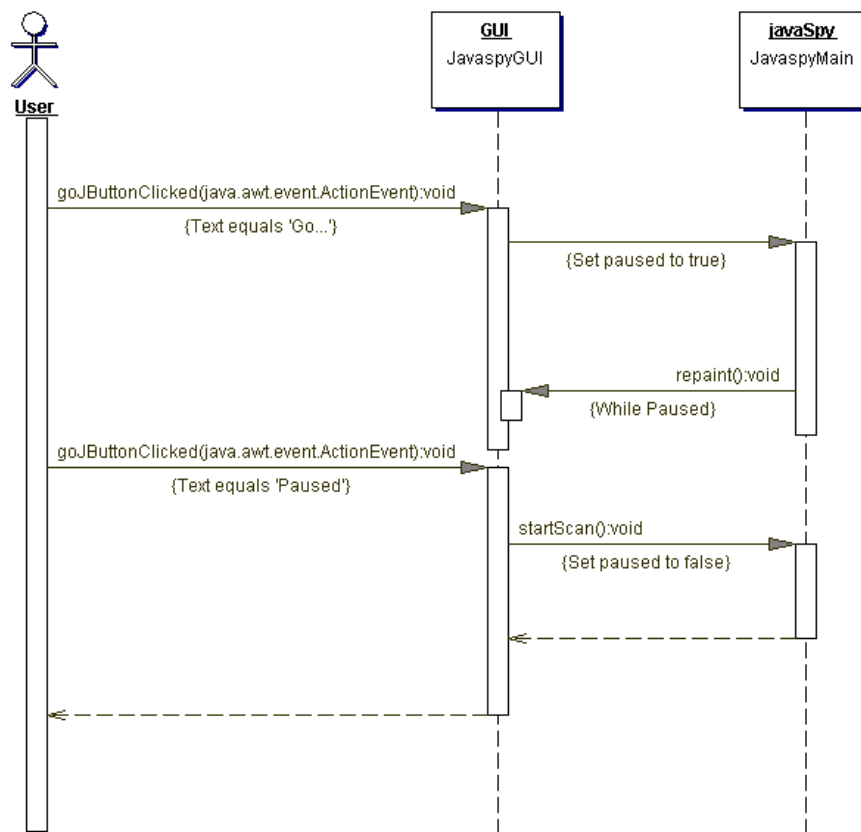


Figure 42: Pause session sequence diagram

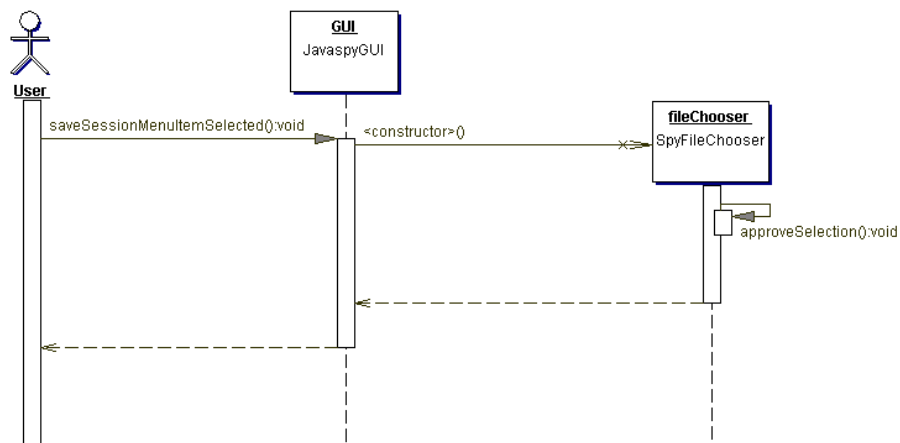


Figure 43: Save session sequence diagram

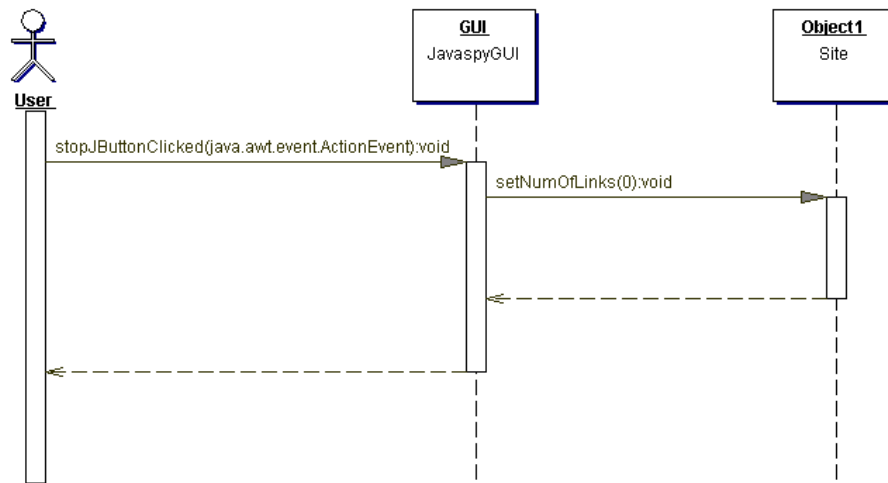


Figure 44: Stop session sequence diagram

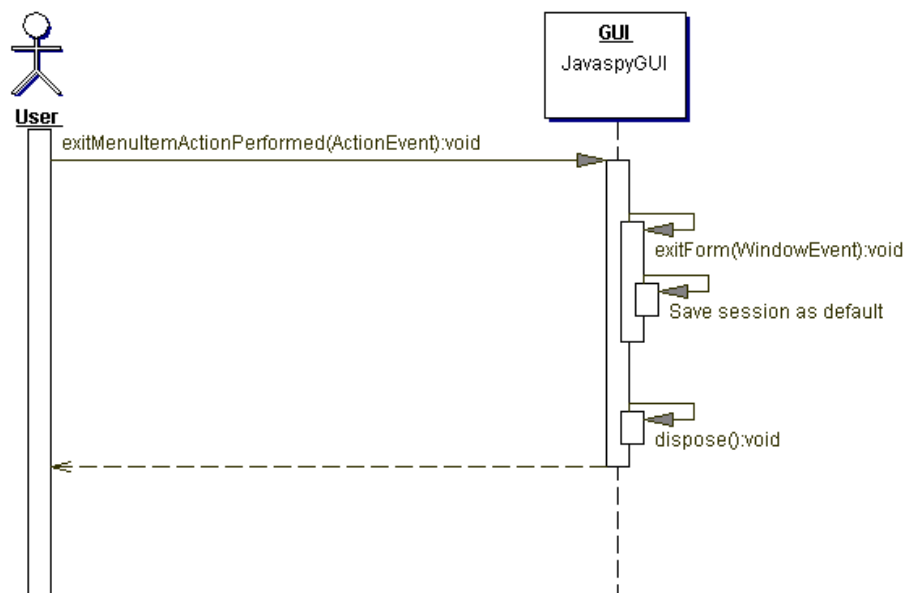


Figure 45: Exit program sequence diagram

Once the sequence diagrams were drawn, I started working on the interface, leaving the core of the system on one side for a while. This allowed me to build the entire interface and have references to the variables and methods available within the interface for information updates on the screen. An activity diagram

describing how JavaSpy works can be found on the next page, as well as the final class diagram.

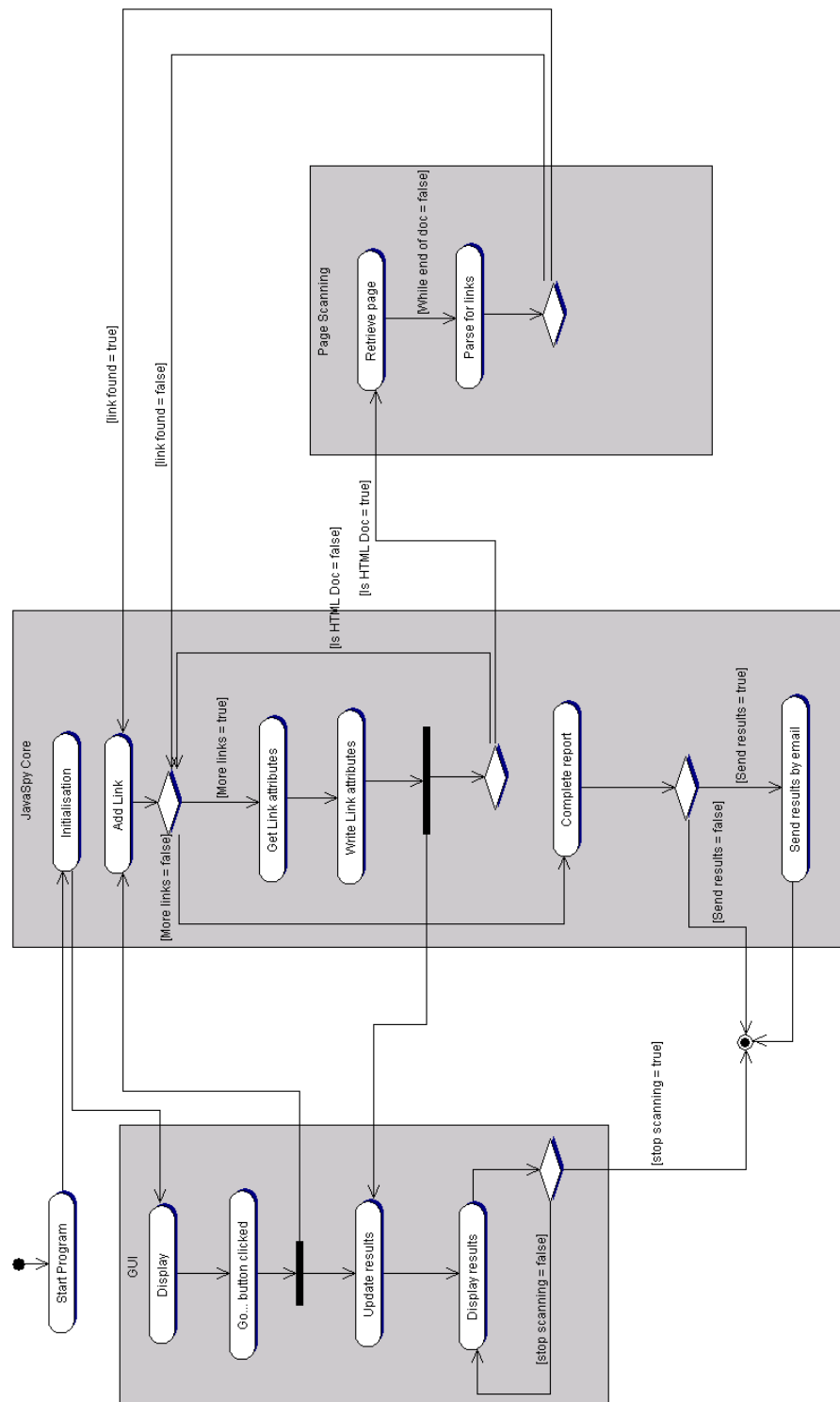


Figure 46: Activity diagram for JavaSpy

13.2.4 User interface

This part of the chapter deals with how tasks that the user is supposed to perform using the system can be described.

The first thing I did was to identify the typical user tasks. For this, task goals will be set and explained, they are shown as a Hierarchical Task Analysis (HTA), and are as follow:

- Start the program
- Edit settings if needed or use default settings or load previously saved settings
- If editing the settings
 - Choose which settings to change
 - Change the settings
 - Apply & close settings dialog box
- Save settings if wanted
 - Give a name to newly saved settings
 - Apply save or cancel
- Once settings are set, launch the scanner
- While scanning, observe information retrieved
- While scanning, pause program
- If program is paused, stop it or carry on with scan
- If program is paused or running, stop it.
- If program has finished scanning, stop it or restart another scan.

This identification of user tasks helps in future HCI research such as seeing how the user interact with the program and understands its functionality.

The second thing I did was to draw a set of diagrams for the tasks that can be accomplished by a user. On the next page is a task diagram for this software. It reflects the points made on this page.

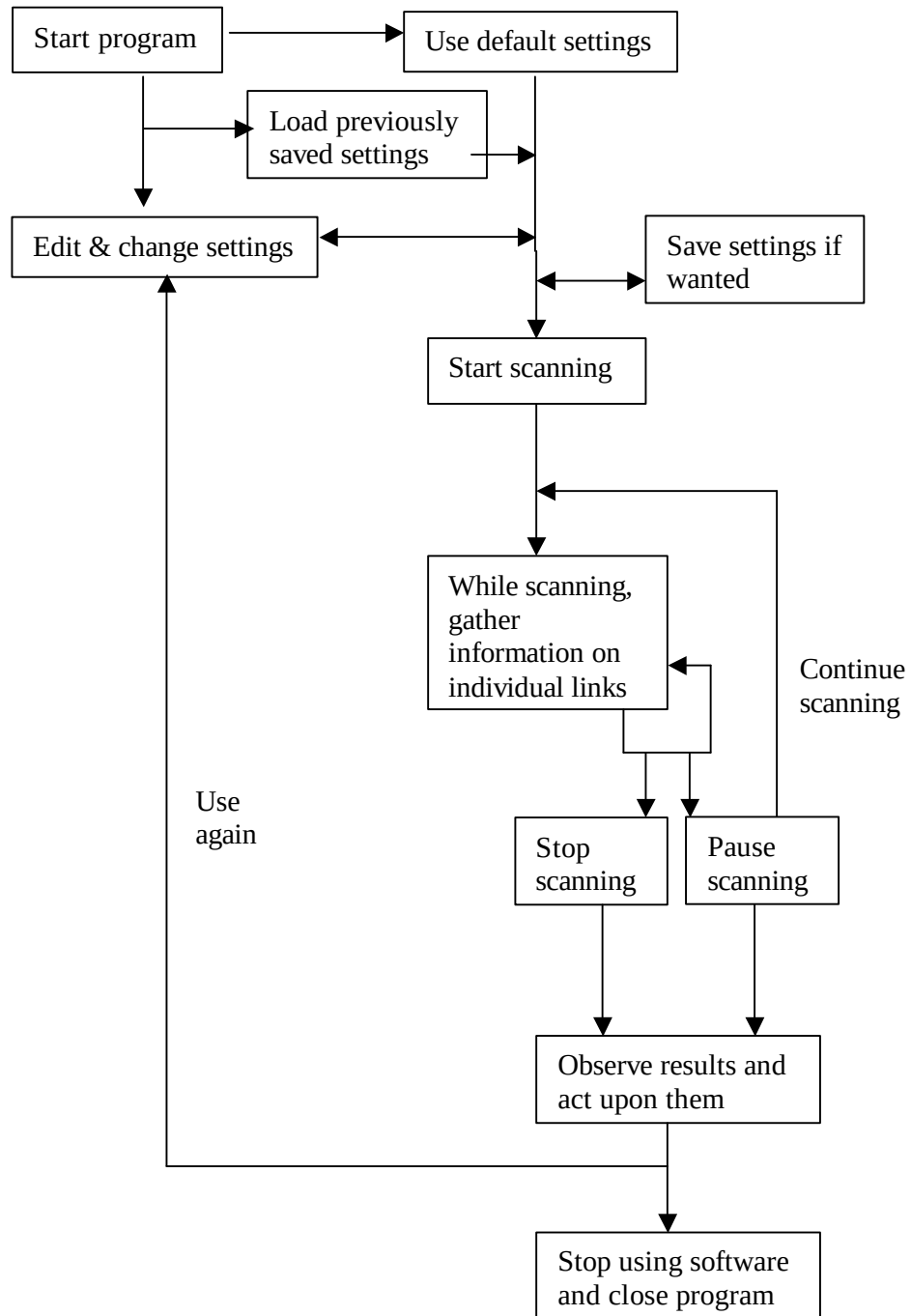


Figure 48: JavaSpy Task Diagram

Below is another diagram that breaks down the editing and changing of settings in the above diagram. The following tasks are therefore sub-tasks of the first diagram.

Once the tasks have been identified, they can be given to the user to see how each task is being accomplished with what degree of ease and if any problems occur during the application of those tasks.

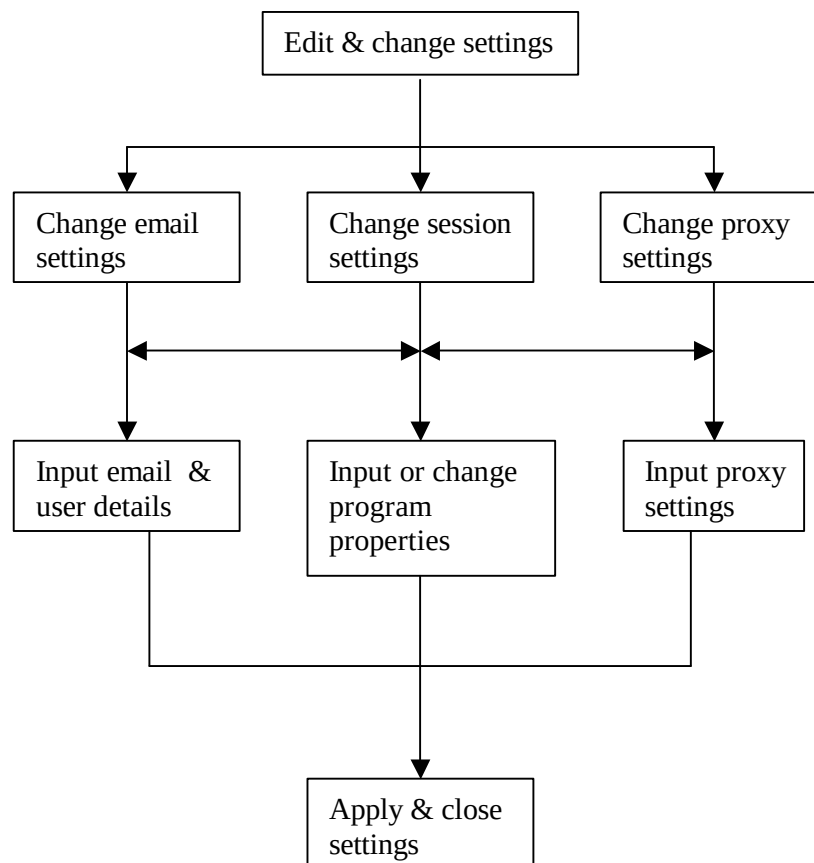


Figure 49: JavaSpy Edit & Change Settings Task Diagram

Designing the interface for the software is crucial if the software is to be easy to use and appeal to a maximum of people. The interface should not be overloaded with irrelevant information, colours used should be justified and anyone looking at the interface should be able to understand quickly what has to be done to accomplish a given task.

The interface built for this software is composed of different areas of interest, below are the different stages of the building process. When the design of each was completed, they were put together to form the final interface.

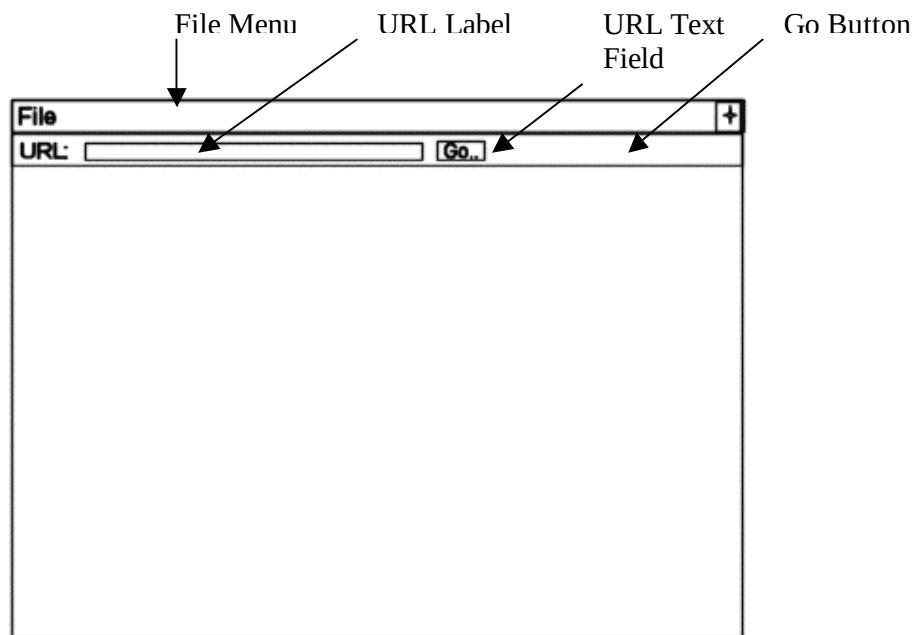


Figure 50: Initial prototype

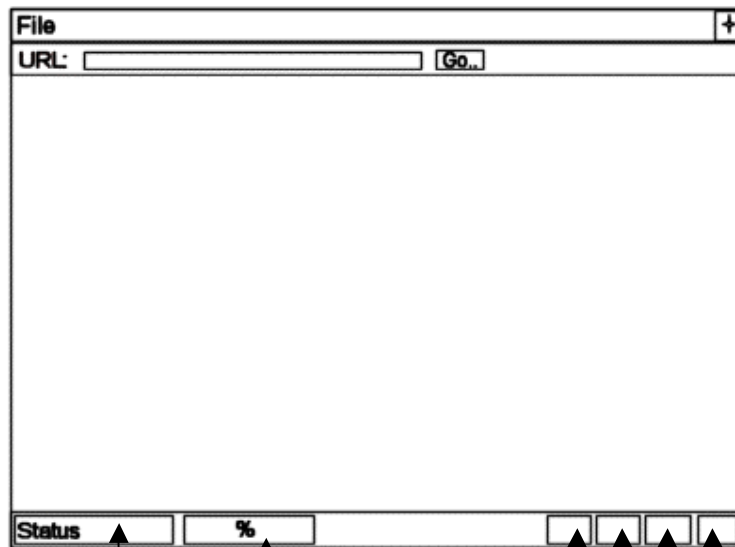


Figure 51: Second Prototype

Program
Status

Percent of
site scanned

Main program properties
reminders

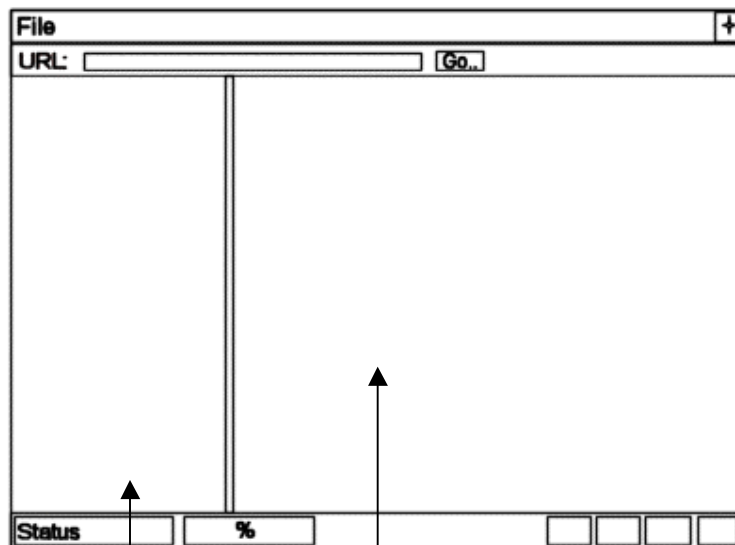


Figure 52: Third Prototype

Information
area

Links area

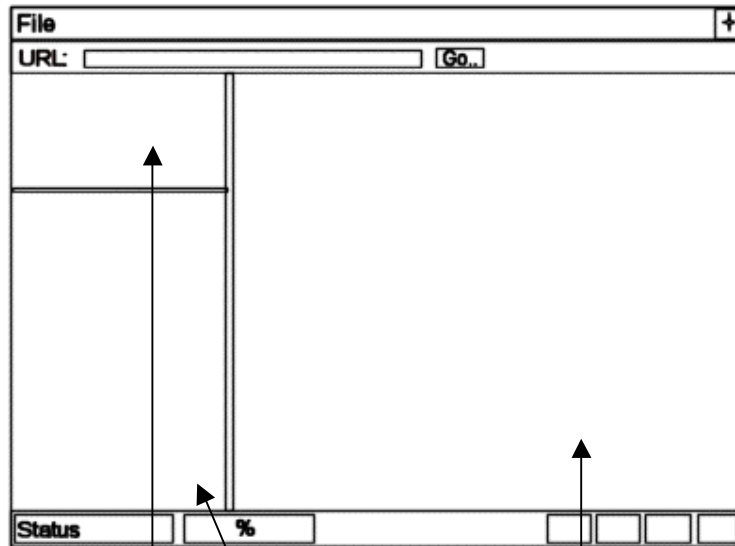


Figure 53: Fourth Prototype

Scanned Site
information area

Individual link
information area

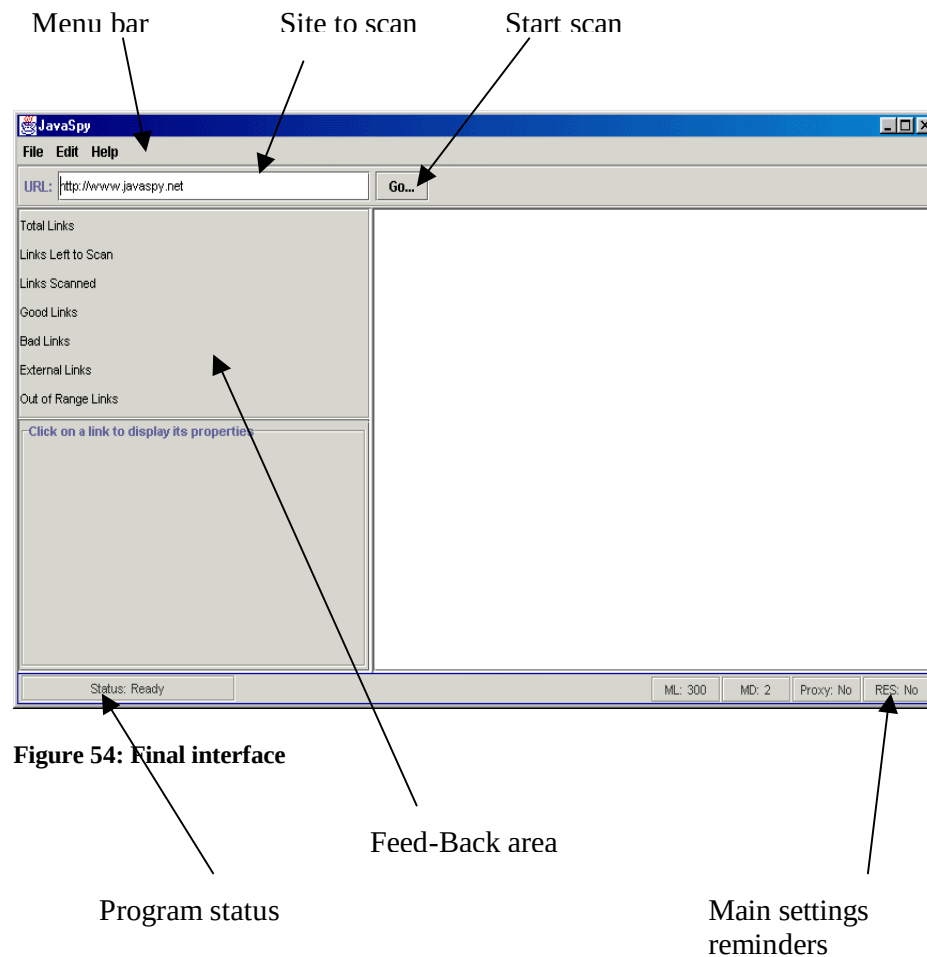
List containing
the links

Once it was decided what the front end of the interface would appear, colour had to be chosen for the background and writings. The list of links displays a vector of links within a web site. There are different types of links and they are listed below.

- Good link (green colour)
- Bad Links (red colour)
- Out of range links (black colour)
- External links (blue colour)

At first, the idea was to display each link name in its own colour, but I realised that if many bad and external links were to be found in the same area of the site, a lot of blue and red lines of text would be mixed and people using this software would most probably strain their eyes looking at the screen. The idea of colour coding the links was important and not to be parted with, so it was decided to use small images at the start of each line, this small coloured surface is not strenuous to the eyes even if many different types of links are mixed on the display. For the window background, I chose a very soft and light yellow colour, the grey used by

most windows program seems to be really morose and sad. The colour is not strenuous and seems to go well with the other colours on the screen. For the file menu, the same colour as the background was used, but when a menu item is selected, its colour changes to a soft purple colour. This colour is standard to most java programs and it was not changed, as it seemed to be most appropriate for the task. Below is the final interface, and on the next page is the interface displaying data as well as a screen shot of the file menu. Hard coding the results has produced these interfaces.



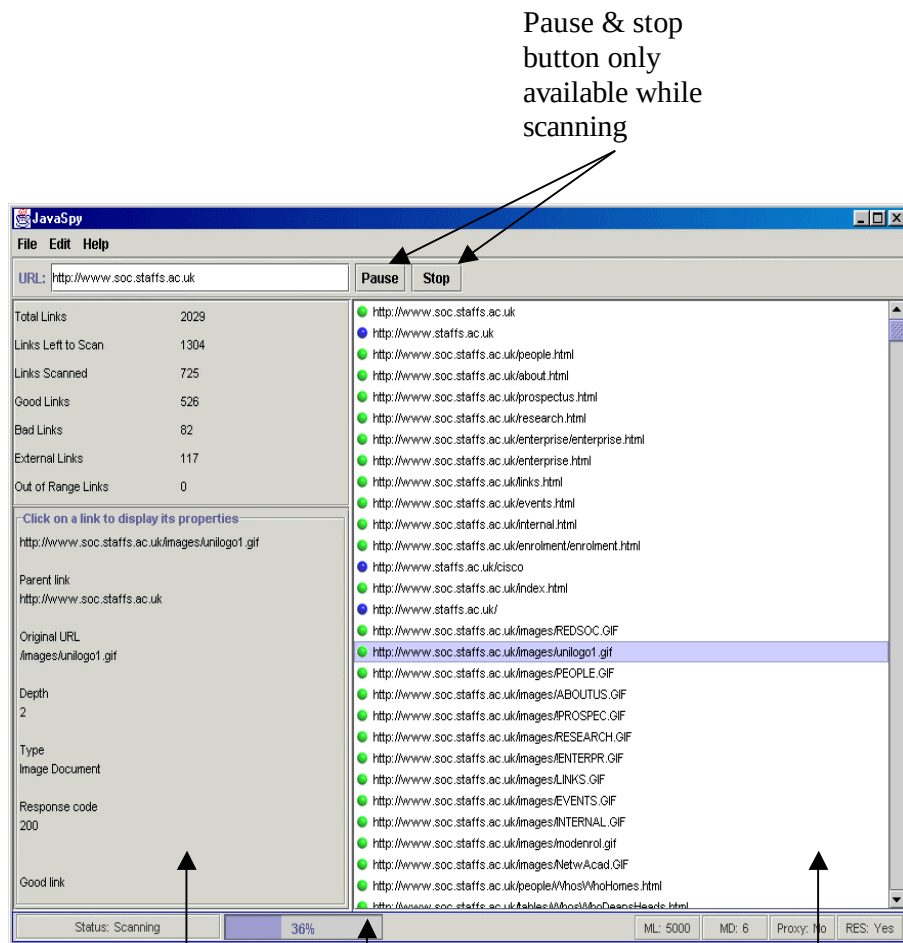


Figure 55: Final interface with data

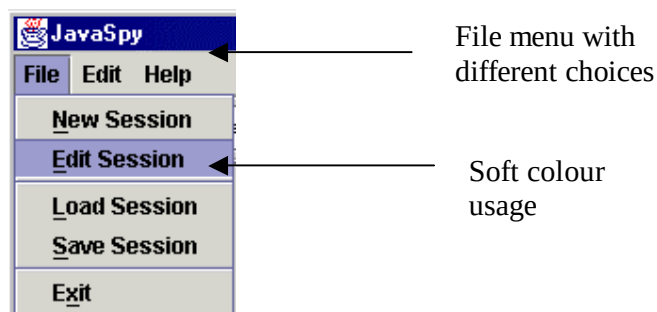


Figure 56: File menu

Another important thing to remember when building an interface, is that users seem to interact better with different bits of information which have things in common being grouped on screen, it makes an interface feel more natural to use. The next few screen shots are of the program properties settings dialog box. This dialog box was built using the same approach as was used for the front end interface, so no prototypes of the dialog box has been added to this document. The main reason for adding the final dialog box is to show how grouping of information can achieve a natural look and feel to the system.

The dialog box comprises three panels, they are used to change settings in the three main parts of the program, and they are: *E-Mail Session Proxy*

Those three panels are discussed in the next pages.

The E-Mail part allows the user to enter his/her name and surname as well as his/her email address and their email server name, so the program may send its scanning results via email.

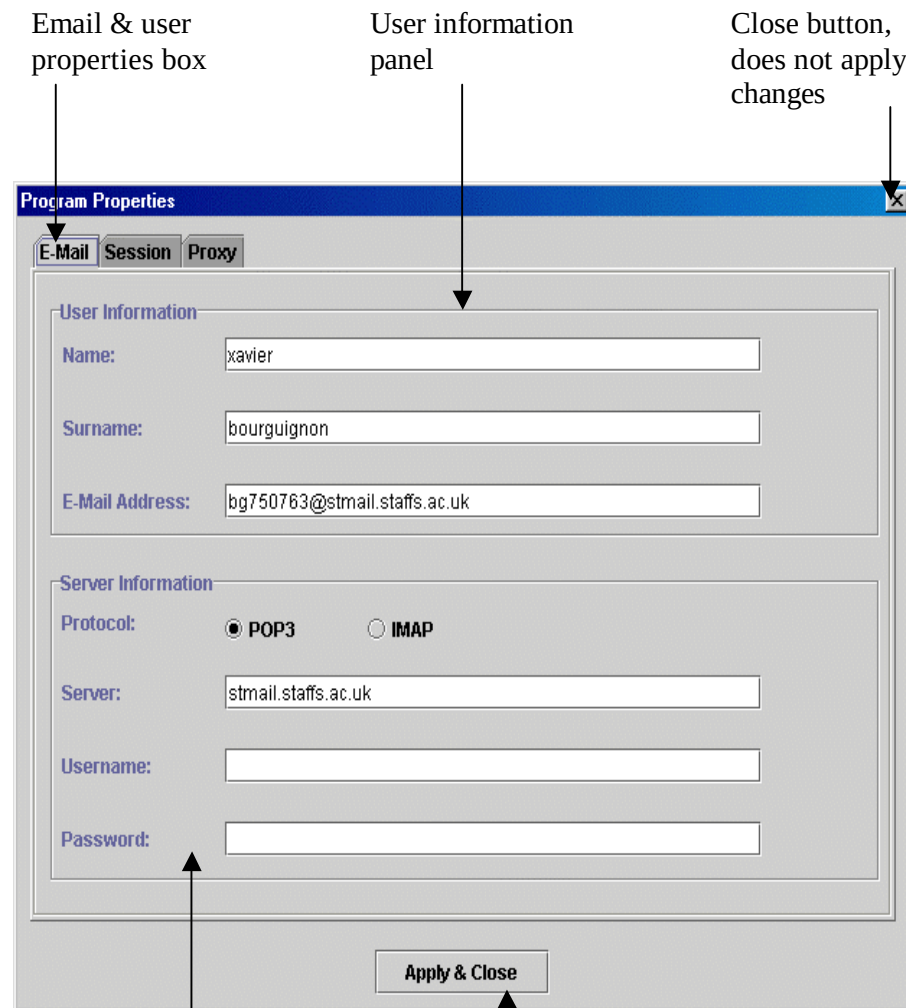


Figure 57: E-mail & user panel

Email server
information panel

Apply changes and close
properties button

As can be seen above, there are two groups, one for the user and one for the email server's settings, it feels clean and easy to use and is self explanatory.

The session part allows the user to change settings for the scanning engine of the program, and also to set certain user preferences. Again, groups have been introduced to facilitate the user's tasks, the first group being for result sending via email, the second being for the scanner settings, and the third for the header that would appear at the top of the result page. Notice that there is a fourth group at the top of the panel containing checkboxes, these define some different functionality with no common ground, but as they are of the same graphical type, they were put together.

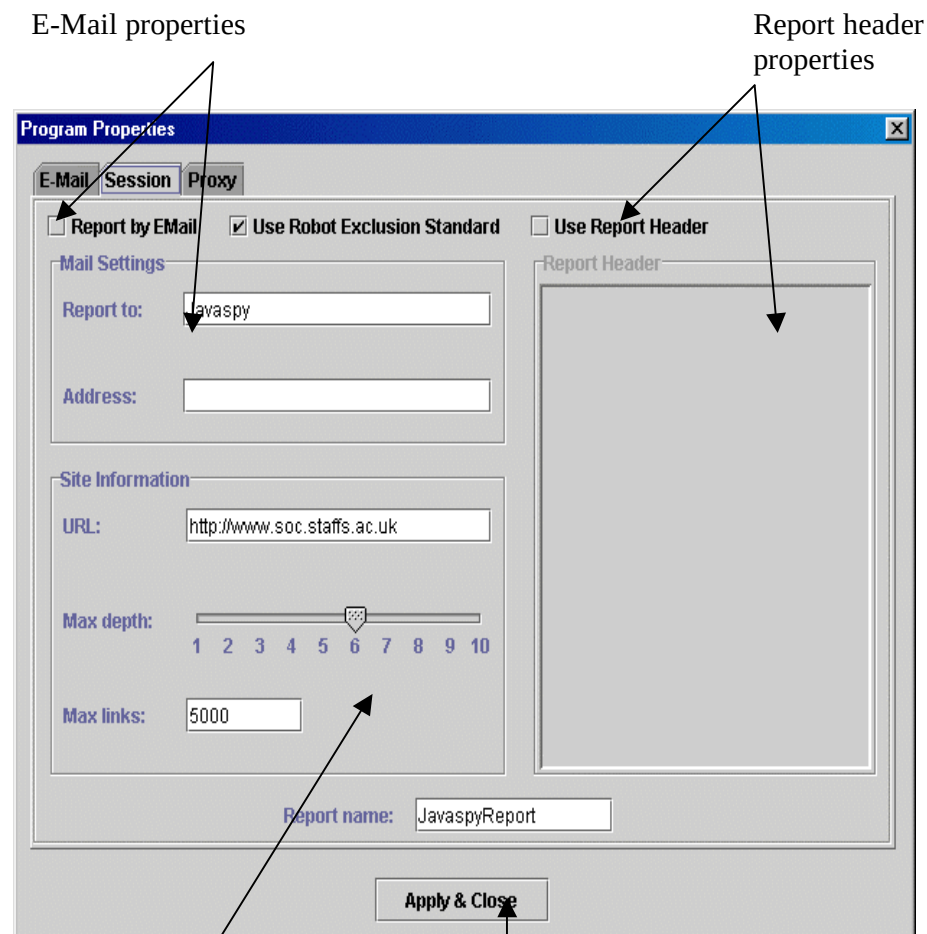


Figure 58: Session panel

Scanning properties

Apply changes and close
properties button

The third part, which is the proxy panel, allows the user to tell the program if it has to go through a proxy server when making connections to Internet resources, and gives the possibility for the user to input a login and password for the proxy server if one is required. Again, different aspects of these settings have been grouped together to allow for an easy use.

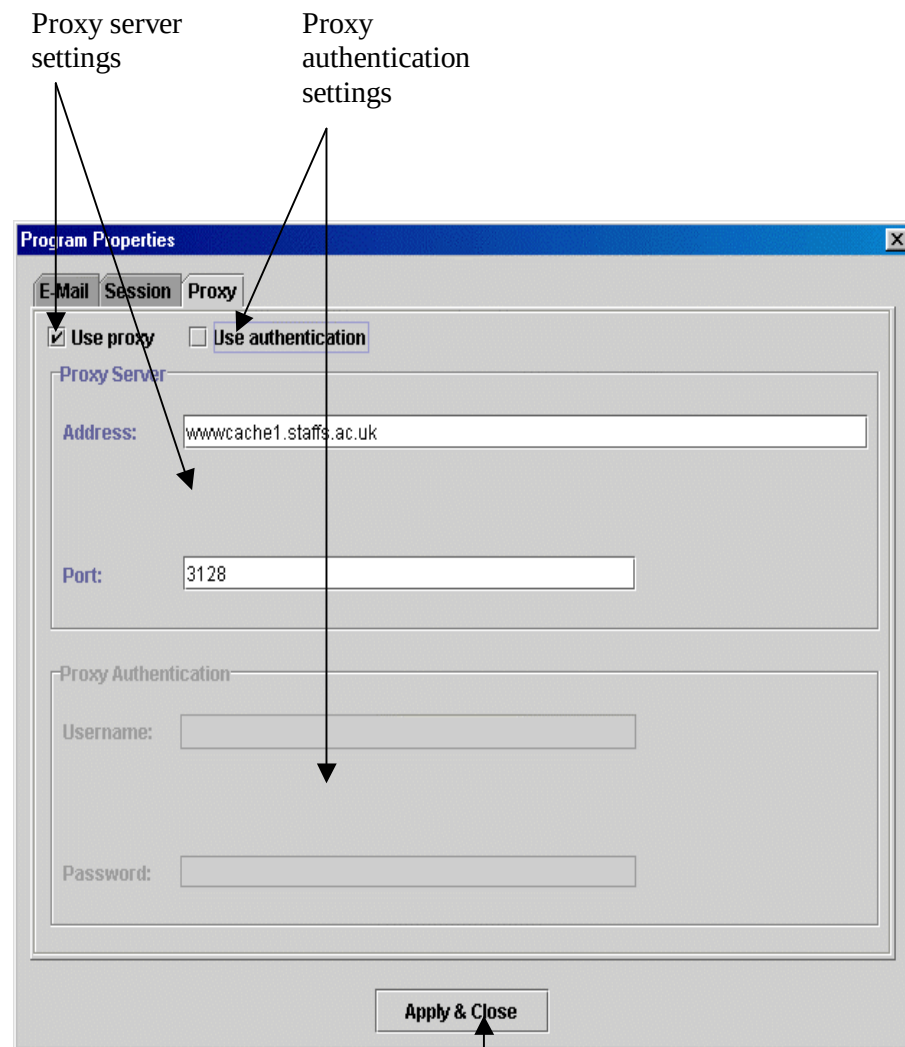


Figure 59: Proxy panel

Apply changes and close
properties button

Once the interface was built, the core functionality was added to the program, the entire source code can be found in appendix B.

13.2.5 User manual

A user manual was written to help anyone who wants to use the system, it can be found in appendix A of this report. Training does not seem to be something that is needed to run this software; most users would be people with their own web site or be web site administrators. These people have a good knowledge of software products and would be able to find their way around the application without any problems.

Testing and evaluation was carried out and details can be found in the next two chapters.

CHAPTER 6

TESTING

14 Testing

In this chapter, JavaSpy will be put to the test by using different testing approaches. These different approaches can all be grouped into what is known as black box testing.

Black box testing tests whether an application actually functions, as it is intended to function. This type of testing is performed by comparing an application's actual functionality with the intended functionality set at design time.

Testing will focus on two parts:

- Functionality test
- Comparison test
- Invalid entry test.

All tests will be done with a maximum amount of links scanned set to 300 and a maximum search depth of 4, this should be sufficient for gathering clear results. All tests unless the ones pointed out, will be done on a machine that connects to the Internet using a LAN with a shared T3 line.

14.1 Functionality tests

1. Check links on a pre-designed site
2. Check links on same site with Robot Exclusion Standard
3. Check links on same site with Proxy server
4. Pause scanning
5. Stop scanning
6. Open new session and change settings then scan
7. Edit current session and change settings then scan
8. Save session
9. Load saved session and scan
10. Load saved session and check that it saved the settings correctly

11. Scan and send result via email
12. Scan a large web site (Microsoft's®) without Robot Exclusion Standard
13. Scan same site with Robot Exclusion Standard
14. Run the program on a machine running under the Linux operating system
15. Run the program on a machine which connects to the internet using a modem and check the speed difference

14.1.1 Results

1. The site scanned is <http://www.javaspy.net>, no external links are present in this site, and there should not be any bad links. There was 172 links found, 109 links were good, 62 were out of range, and 1 was bad. This bad link was not expected and I checked it manually and indeed this link was pointing to a non-existent resource.
2. A small part of the site has been disallowed to all robots by putting a *robots.txt* file in the root directory of the site. There was 159 links found, 96 were good, 62 were out of range and 1 was bad. JavaSpy did not attempt to make any connections to the disallowed part.
3. Proxy settings were entered and the scan launched. The scan returned the same results as test 2, but the results were quicker to come.
4. During scanning, the pause button was clicked, and JavaSpy paused scanning as expected, the go... button was clicked and the scan carried on. This was done three times and every time JavaSpy responded perfectly well.
5. During scanning, the stop button was clicked, JavaSpy stopped. There seems to be a small problem when using the stop button, if a connection is underway, and the response from the scanned site is very slow due to traffic on the network, it may take a small amount of time for JavaSpy to stop its scanning as it waits for a connection to be terminated before it does so.
6. A new session was opened. Settings were changed at random and a scan was done with the new settings. JavaSpy scanned the site using the new settings without any unexpected results.

7. The previous session was stopped, and then edited. The settings were changed and a scan was done. Results expected appeared on screen.
8. After editing and changing some settings, the session was saved as *javaspy.spy*
9. JavaSpy was closed and re-opened. The *javaspy.spy* file was loaded and a scan was started. The settings had been saved properly; this was checked by editing them and making sure that they were the same as when they were saved.
10. See test nine.
11. A scan was done and when all the results were in, they were sent via email to the email address present in the settings. Those results were checked against the ones on screen and were identical.
12. A scan of Microsoft's® site was done, results seemed to come through rather slowly, so I went to Microsoft's® site with a browser, and indeed it was very slow.
13. Once the scan started, completely different results came through, this was due to the fact that JavaSpy was not scanning certain parts of the site. To make sure this was right, I downloaded Microsoft's® *robots.txt* file and checked to see which parts were disallowed, this matched the results previously obtained.
14. JavaSpy was run under the Mandrake 7.1 distribution of the Linux operating system. The java virtual machine was version 1.3 from Sun Microsystems® which uses native threads. Using native threads can make a difference to an application's speed of execution, but for this can only be achieved with very large software systems built in java. The program ran fine, and acted in exactly the same way as if it was running under Windows®. The interface looked a little different as JavaSpy uses the operating system's look & feel for displaying interface components.
15. A scan was done on a machine with a 56k modem. The scan seemed to be more fluid than on a LAN connection, this is most probably due to the fact

that the LAN connection is shared and that requests may take a relatively long time before being sent to a web server.

14.2 Comparison test

1. Scan a site with a similar product then with JavaSpy and compare results.

14.2.1 Comparison test result

1. A site was scanned with an application called LinkBot. It returned the same results as JavaSpy did, but was much faster than JavaSpy in scanning the site. JavaSpy took about 2 minutes as LinkBot took half the time. LinkBot used twenty threaded connections and this is most probably why it was faster.

14.3 Invalid entry test

1. Set proxy on without entering settings
2. Set proxy on with erroneous server settings
3. Scan a non-existent site
4. Create a file with .spy extension and try to load it.

14.3.1 Invalid entry test results

1. JavaSpy was set to use a proxy server to connect to the Internet, but no data was entered for the server. As expected, an error message came up explaining what could have gone wrong.

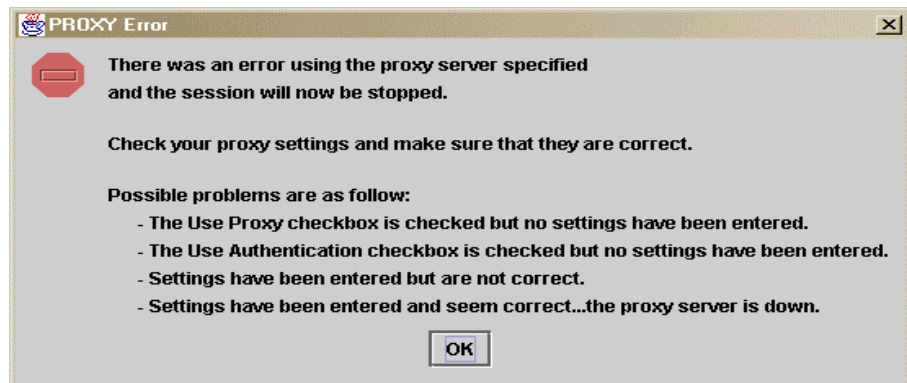


Figure 60: PROXY error message

2. JavaSpy was set to use a proxy server, but a non-existent server name was entered. As expected, an error message came up, see figure 1.
3. The following URL was entered in JavaSpy: *www.soc.staf.ac.uk*. This site does not exist and JavaSpy came up with an error see figure 61

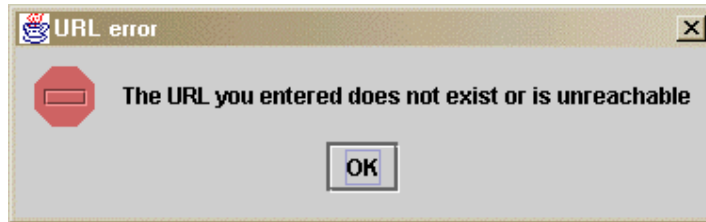


Figure 61: URL Error message

4. A file with *.spy* extension was created manually, and then loaded in JavaSpy. An error message came up, this message can be found below.

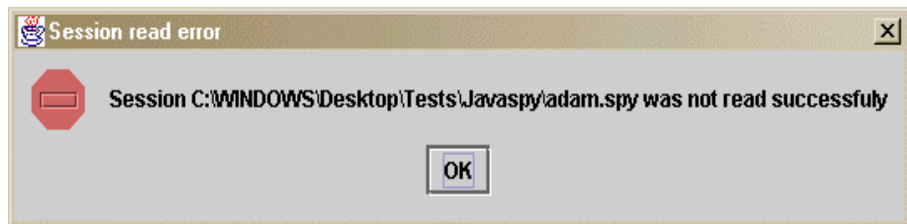


Figure 62: Session read error

These tests have shown that JavaSpy does what it is supposed to do. It seems to be able to handle large site as well as small ones. The speed at which it checks links can be disappointing if running of a very busy network. Many more features could be added to the program and will probably be in the future.

CHAPTER 7

EVALUATION

15 Evaluation

Evaluation will be based on the user interface and not on the program's functionality as tests have been carried out to check that the program does what it is supposed to do. It has been decided that interviewing people for this evaluation would take too much time out of an already tight schedule. Therefore, a questionnaire will be given to a few people who are going to evaluate the user interface and its functionality.

Evaluating an interface is important to find out any problems end-users may encounter while using it. There are different types of evaluation techniques available, these techniques fall into two main groups, expert evaluation using well-known HCI methods, and evaluation based on the user, with the user. Expert evaluation forms the first part of this chapter; the second part will involve evaluating the interface with users.

15.1 Expert evaluation

Expert evaluation means that HCI expert(s) using well known methods, will evaluate any kind of interface to a product, it could be a software product but it can also mean any kind of hardware used to accomplish given tasks. There are different kinds of methods available to an evaluator, one of them will be discussed and used on the final interface to find out if it is worthy of being distributed or if any major changes need to be done before its release.

15.1.1 Heuristic evaluation

Heuristic evaluation can enable many usability improvements to take place before a release deadline that would not permit usability testing. Research carried out in the HCI community shows that such evaluations can identify a majority of the usability problems. For this purpose, I will be the only one evaluator of the interface, and being the person who built the interface, the results may not be as subjective as they should be, but a serious and professional approach should solve this problem.

The major drawback of heuristic evaluation is that any evaluator, regardless of his/her skill and experience, remain a substitute user (someone who emulates a user) and not necessarily a typical user of the product. The results of heuristic evaluation are not actual user data and therefore should not receive as much credit as results from studies with actual users.

Real users often surprise expert evaluators, they often have problems that were not expected, and sometimes breeze through where they were expected to fail or get stuck. Other reasons why heuristic evaluation shouldn't replace studying actual users are that it rarely emulates all the key audience groups, and it doesn't necessarily indicate which problems users will encounter most frequently.

Heuristic evaluation usually explore the following questions:

- How simple to use is the interface? For a more complex task, how well does the interface step the user through subtasks?
- How clear are the meanings of graphical elements such as icons and toolbar buttons? Are they overused or underused?
- How well is the interface organized? Are navigational aids adequate to support the organization? What feedback is provided to orient the user?
- Are instructions or explanations presented clearly, without unnecessary complication or ambiguity? Is the language direct, simple, and non-wordy, so that users can read/hear as few words as possible to accomplish a task?
- How effectively are analysis and/or search results presented on the screen? What window manipulations are required to view results easily?
- What information (text, voice, or graphics) must users encounter that they don't need? What information might be missing?
- How well does the interface assist users in recovering from problems?

These seven questions will now be applied to the interface and answered as truthfully as possible.

- The interface is simple to use with well laid out components, labels are meaningful and colours used are appropriate. When achieving a long task

such as changing settings, the interface remains easily used and understood.

- Graphical elements are basic and only used when necessary. Their meaning are easy to understand
- The interface is organised in such a way that every task, which has to be accomplished, is done so, easily. Components are grouped by task relevancy, and do not require excessive use of the keyboard or mouse.
- A problem this interface may face is the lack of explanations or instructions to the user. It was felt that the interface and the program it belongs to are made for a single line of events and help to solve one predefined problem. For this reason, a help menu was not built but instead a user manual was put online for anyone to look at and gain information, which could be of importance to the user if necessary.
- Search results are presented on a predefined area of the screen as soon as they are available. The software can be understood as being a scanner of some kind, information keeps piling up as long as scanning is in progress, this information is displayed in a list like view. Scroll bars are displayed as necessary when there is more information than can be fitted on the screen, this allows the user to look at a given amount of information at any one time, therefore avoiding information overload which often occurs with badly designed interfaces.
- Information is given to the user only when necessary, important program settings displayed discreetly at the bottom right of the screen. More information could be given, and much more functionality could be added to the program, but the interface seems well suited for the tasks users would have to accomplish using this software.
- If an error occurs due to user error or program error, the user is alerted immediately and an explanation is given of what the problem may be. In case of a user error, the program displays information as to what happened and what can be done to solve the problem, therefore giving the user closure on to what is happening and what has to be done.

15.2 Evaluation with the user

Evaluation of interfaces with users is the most meaningful way of seeing and understanding how users will interact with the product. For this, I have asked four people to help me and they agreed to give a little of their time to evaluate the interface.

15.2.1 User characteristics

Knowing about the user is a critical stage in the development of any software systems that are meant to be used by people, and one of the best ways of gathering such data is to write a questionnaire and have it answered by users of the future system's interface. As mentioned in previous chapters, the system should be able to be used by anyone, although the people who are most likely to use it are web site administrators or people who have a website, which often implies that the user will be computer literate to a certain extent.

Below is a questionnaire that was built to try to identify who would use such software and their level of experience with computers. The questionnaire also tries to find out any possible handicap a user may have when using the interface. The questionnaire can be found below.

All questions marked with a * have no choice accompanying them, please write a short and meaningful answer.

- a) *What is your job?
- b) Do you often use computers? *Never Sometimes Regularly Always*
- c) Where do you mostly use a computer? *At home At work Public places*
- d) Do you consider yourself to be an experienced user of computers? *Yes No*
- e) What sort of software do you use? *Games Development Office work*
- f) Are you familiar with the Internet? *No A little Yes Use it all the time*
- g) *If you are familiar with the Internet, describe what use you make of it.
- h) If you have a web site, who built it? *Third party My team Myself*
- i) If you have not got a web site, would you like to build one? *Yes No*
- j) Are you: *Female Male?*

- k) Are you: *Right handed* *Left handed* *Ambidextrous*?
- l) *What is your highest qualification?
- m) Have you got any problems using a keyboard, a mouse or any other kind of computer related controllers? *Yes* *No*
- n) *If you answered yes to the above question, can you describe what the problems are?
- o) Are you colour-blind? *No* *Yes (please describe)*

This questionnaire could probably be more intense and contain many more questions, but it seems sufficient to ascertain the most important characteristic of a future user of the interface.

Another way to gather user information is to have the user use the prototype interface and observe how it is being used. A lot may be learnt by gathering information in this way, by looking at the user's reactions to events on the computer monitor, and also by trying to determine the length of time it takes the user to get familiarised with the interface.

Once a user has utilised the new interface, another questionnaire can be handed out to find out more technical information on the likes and dislikes of the user towards the interface.

Two of the four evaluators have their own web site, the third one uses computers a lot and knows the Internet pretty well, the fourth candidate rarely uses computers but has a rather good understanding of what has to be done. In this report, they will be known as candidate 1 to 4.

It was very important to have those people evaluate the interface each in the same conditions, so it was decided that they would evaluate it on my computer at home.

Each candidate was explained what the software was about and what help it would bring a web site administrator. The three people with the most computer knowledge had no problems understanding what was asked of them, and understood what the software was about. The third person understood what was asked too but had to be explained a little what the actual software would achieve in the real world. The idea behind the user evaluation is to observe the users while they are using the software and try to determine if any problems occur or if they

have any problems understanding what something does. The result of my observations can be found below. Characteristics about each user were found by asking them to answer the questionnaire present on the two previous pages.

15.2.2 Interface evaluation with the user

1. The first candidate was the person with the less knowledge about computers. Her first remark when seating in front of the screen was: *It looks simple, I thought it would have buttons and menus everywhere.* I asked the person to click on the file menu and choose New Session. When the dialog box appeared, it was felt that the candidate was a little lost as what to do next, I explained the role of each panel and had to tell the candidate what to input in each field. Before applying the settings and closing the dialog box, I asked if the interface looked fine in terms of colours and positions of the components, the candidate answered positively but I could feel that the person was not really used to utilise this kind of software. Once the dialog box was closed, I asked the user to start the scanning by clicking on the 'go...' button and to wait for results to appear. While the software was running, the user could have pressed any button or even click on displayed information to extract additional knowledge about it, but she remained there looking at the screen without attempting any interaction.

I felt that this evaluation was going rather badly and wondered if it was a problem with the interface or a problem with the user. This was going to be answered with the next user's evaluation.

2. The second candidate who has her own web site acted in a completely different manner to the first candidate. She also mentioned how simple the interface looked, but with a different approach, she thought it would not be able to do what it was suppose to do. I asked the person to open a New Session via the file menu and to fill in the details. It took candidate two a very short amount of time to see what was needed and all the settings were ready in under a minute. Before applying changes and closing the dialog

box, I asked the same question about the layout and colours used for the settings dialog box, the response was very positive, with words such as: easy to understand, well separated (groups) areas, simple. I let the user carry on and did not have to tell her anything about what to do next. During the scanning, the candidate paused the scanning several times, clicked on displayed information while scanning and while paused, and tried to open the file menu. The candidate stopped the scanning and restarted it with a bogus URL; the interface displayed an error message explaining that the URL entered did not exist. Eventually the user stopped and said that the software looked ok, and that the presence of a progress bar at the bottom of the interface was most welcome, and that it was good for knowing approximately how much time was left to the end of a scan.

It was felt that this evaluation went much better than the first one and that the user's knowledge about computers and the Internet was a great plus.

3. The third candidate also has a web site, he also programs a lot and knows about interface and how difficult they are to produce. It must be pointed out that this candidate had seen the interface before and seen me using it, but he had never used it himself. I let the candidate alone and after 1 or 2 minutes, the software was scanning his site for broken links. The main comment from this user was that he was especially impressed with the layout of the components in each panel in the settings dialog box. This person programs in Java and knows how difficult it can get to produce a good interface in this language. One comment was that the dialog box should have a cancel Button, an Apply button, and an OK button instead of a single button called Apply & Close. Once the scanning was finished, the candidate asked me if he could "break" the software, I answered that he could try. Breaking a software means that a user will try anything to make the software crash or act, as it was not supposed to. Candidate three tried his best and I must say that eventually he found a flaw in the system. This problem had not much to do with the interface but was important enough to mention here. When a user clicks on the Stop button during a

scan, the program stops scanning and the Go button appears again. If the user tries to click on the Go button, nothing happens, the user must first edit a new Session or the existing Session before being able to carry on. It was mentioned that the button label should say something different to avoid confusion with new users. While candidate 3 was trying to “break” the software, several error messages came on screen explaining what was wrong and some explaining what to do to solve a particular problem, I asked if these error messages were appropriate, the answer I got was: *It makes a change from a message that says: error 102, contact the vendor.* This remark was funny but very important to me as I felt that closure was achieved with the user when a problem occurred and that the user was not left stranded with an unsolvable problem.

4. The fourth candidate was not able to spend much time evaluating the interface and I could not get as much information as I would have liked. Candidate 4 seemed to be pleased with the colours used and the layout, a quick scan of a small web site was done and once it was finished, the user said that it would be a good idea to add a functionality were the program could sort each links on the screen by types, this would avoid having to scroll up or down too much to find out what links had an unsuccessful connection. I thought that is was a good idea and have yet to implement this feature.

Overall, user evaluation went well. It is obvious that this software is for people with computer and Internet knowledge and that someone who does not posses this knowledge would have great difficulties using the product. It was probably a mistake to use candidate 1 in this evaluation, but experience was gained and future evaluations will not involve users for which the product is not intended.

15.2.3 User satisfaction questionnaire

A post evaluation questionnaire was made and given to each candidate to answer.

Results were collected and calculated.

The questionnaire can be found below.

1. The interface layout was:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

2. The interface usage of colours was:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

3. How easy was it to accomplish given tasks:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

4. The length of time it took to get familiarised with the interface was:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

5. Display of results on screen were:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

6. Responsiveness of the interface was:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

7. How easy were the error messages to understand (if any came up):

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I
10 9 8 7 6 5 4 3 2 1 N/A

8. How would you judge this interface overall:

OUTSTANDING *OKAY* *UNACCEPTABLE*
I-----I

10	9	8	7	6	5	4	3	2	1	N/A
----	---	---	---	---	---	---	---	---	---	-----

Results for this questionnaire were as follow:

Overall, the interface got a usability score of 7.4 out of 10. It has to be mentioned that in industry, many more people would be involved in evaluating an interface, experienced evaluators working as a team with a large number of end-users to test with. This result is only relative to the number of people used, and human factors (such as knowing the people involved) may have got in the way of this evaluation, although it was stated to the candidates that they had to evaluate JavaSpy as seriously and professionally as possible.

Deciding what software to design and build and what method to use for such a task was the most difficult part in this project. The subject had to be interesting, with sufficient background research, analysis and design scope so it would form a complete piece of work which would gain a high mark for each sections, but also not too complex that it would not have been possible to finish in time. I feel that

some of the work carried out during my industrial placement gave me some knowledge on the programming side of things, but unfortunately I did not gain any knowledge on how to produce, design and implement a full system.

I am satisfied with the work carried out during this project. All the features I wanted to incorporate into the program were done successfully. The research carried out helped me in understanding and planning what was required for such a system, and also increased my knowledge in a lot of areas.

15.3.1 Problems encountered during this project

The biggest problem I encountered during this project was to decide what objects would be needed during the design stage. Such was the problem that if a prototyping approach had not been used I would have not been able to produce the program as it is now. This problem is due to the fact that this was my first full project design, and experience was lacking in this most important stage.

Another problem was to design the user interface. Choosing carefully the colours and the placement of graphical components took some time, and this was tackled by taking a trial and error approach.

One other small problem occurred during the implementation. I always make sure that any software I use is set to auto save, and saves my work every one minute. Unfortunately, one day I forgot to check this feature and a power cut occurred, this resulted in the loss of some source code in one of the objects. This was not too much of a problem, and I only ended up losing about two hours of work.

15.3.2 Lessons learnt

The most important lesson I learnt was the time management side of the project. I realised once most of the work was done that if I had not followed the gant chart exactly, I would have had problems. Sticking to the time schedule was very important and I am glad that I did so.

Another important lesson I learnt was testing source code as soon as it is written and when possible. This allowed me to have very little debugging to go through at the end of the implementation stage.

I also learnt that taking time to do the research and understanding all of the research gave me a lot of knowledge to carry on with the other stages of the project.

Finally, one of the thing, which I already knew about and is really hard to implement, is to know when to take time out, stop working and relax a little. I have seen many people struggling during their projects because they were at it for too long and started losing focus on what they were doing, I am glad that previous experience had taught me this lesson already.

15.3.3 Things I would have done differently

Looking back at the project, I would have changed a few things in the way JavaSpy works if I had to do it again. I would multi thread the connections made to the Internet to make the scanning faster, although it would have to be implemented correctly so not to overload the server from where the information is coming from.

I would also have liked to implement an HTML viewer so the results of a scan that are kept in an html file could have been displayed directly using JavaSpy itself and not a commercial browser.

Another thing would have been to implement a JavaScript parser, so links that are hidden within JavaScript could have been checked for, but this in itself is like another project and the time it would have taken would have been too great.

15.3.4 Conclusion

I feel that going through the process of researching, designing and implementing a fully working system has given me the experience needed to enter the world of work in computing. I learnt a lot about system design and HCI, and I feel that the project modules were the best modules I took during those four years spent at Staffordshire University. These four modules combined together were the most interesting and gave me more knowledge than any other module.

CHAPTER 8

REFERENCES

Books

Harold Thimbleby. (1990). User Interface Design, New York, ACM press.

Jenny Preece and Laurie Keller. (1990). Human Computer Interaction, London, Prentice Hall.

Ben Shneiderman. (1987). Designing the user interface: Strategies for effective human-computer interaction, Wokingham England, Addison-Wesley publishing company.

Donald A. Norman and Stephen W. Draper. (1986). User Centered System Design: New perspective on human-computer interaction, London, Lawrence Erlbaum Assoc.

Ian S. Graham. (1995). HTML Sourcebook, USA, John Wiley and Sons, Inc.

Molly E. Holzschlag. (1998). Using HTML 4 special edition, USA, QUE Corporation.

Thomas A. Powell. (2000). HTML: The complete reference, London, Mc Graw-Hill professional publishing.

Martin Fowler. (1997). UML distilled, Harlow England, Addison-Wesley publishing company.

Joseph Schmuller. (1999). Teach yourself UML in 24 hours, Indianapolis, Macmillan Computer Publishing.

Joseph L. Weber. (1998). Using Java 1.2, USA, QUE Corporation.

Merlin and Conrad Hughes, Michael Shoffner, Maria Winslow. (1997). JAVA Network Programming, Greenwich, Manning Publications Co.

David Flanagan. (1997). JAVA in a nutshell, Cambridge, O'Reilly & Associates, Inc.

David Flanagan. (1997). JAVA Examples in a nutshell, Cambridge, O'Reilly & Associates, Inc.

Ellis Horowitz, Sartaj Sahni, Susan Anderson-Freed. (1993). Fundamentals of Data Structures in C, London, Computer Science Press.

Internet

<http://www.bcs-hci.org.uk/> accessed on 26th October 2000

<http://www.ida.liu.se/~miker/hci/guidelines.html>. Accessed on 5th November 2000

<http://www.w3.org/>. Accessed on 20th November 2000

<http://werbach.com/barebones/>. Accessed on 20th November 2000

<http://www.hwg.org/>. Accessed on 21st November 2000

<http://www.platinum.com/corp/uml/uml.htm>. Accessed on 12th December 2000

<http://www.robotstxt.org/wc/exclusion.html>. Accessed on 22nd January 2001

<http://www.robotstxt.org/wc/robots.html>. Accessed on 22nd January 2001

News groups

comp.human-factors. Used on 26th October 2000

comp.graphics.visualization. Used on 28th October 2000

APPENDIX A

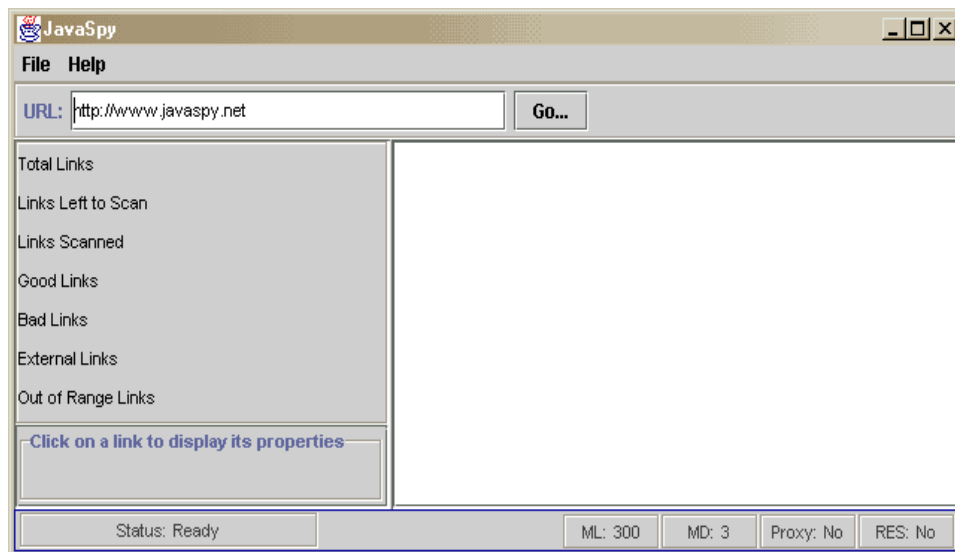
USER MANUAL

User manual for JavaSpy

Prior to using JavaSpy, you must have the JDK 1.3 installed or the Java Runtime Environment 1.3 installed. If you have the development kit, the PATH and CLASSPATH environment variables must be set correctly (refer to sun's documentation on how to do so). The CLASSPATH variable must also contain the working directory, which can be set by adding a dot (.) in the CLASSPATH variable's declaration. The JavaMail API should also be present to allow JavaSpy to send results via email; the JavaMail API is small and only takes a few minutes to download with a modem connection. If you do not have JavaMail, you may find it on Sun Microsystems® web site along with the documentation on how to install it, it is free for use.

Unzip JavaSpy into a destination folder of your choice. To start the program, simply double click on the *spy.bat* file.

JavaSpy has been designed with ease of use in mind. The interface is simple and clear as shown in figure 1. JavaSpy's functionality depends on a session. A session is what tells the program what to do and how to do it. The default session which is loaded at the start of the program is very basic, it has a default URL set to JavaSpy's web site, with a maximum depth search of 3 and a maximum link search of 30. The default session does not use a proxy server to connect to the Internet. Every agents that makes rapid fire connections to a web server should obey the RES (Robot Exclusion Standard), this standard was designed to allow web site administrators to block entry to parts of their site or the entire site. JavaSpy implements this feature, and it is turned on by default, it may be turned off, but this could mean having very unhappy site administrator who could eventually block any agent's entry to their entire site. It is therefore recommended to use the RES. On the next pages, you will be shown how to operate JavaSpy and how to open a new session, edit a session, save a session and load a session.



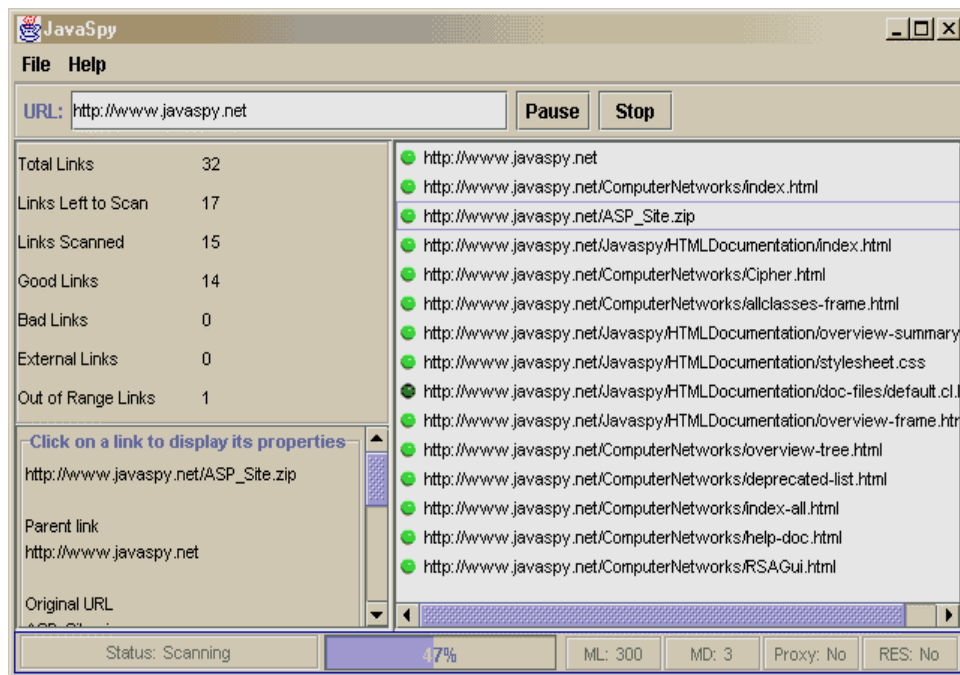
JavaSpy's status bar gives information about the running of the program. It is situated at the bottom of the window. On the left is the program's status. The status may be any of the following:

- Ready (The program is ready to scan a site)
- Scanning (The program is scanning a site)
- Paused (The program has been paused)
- Stopping (The program is in the process of stopping its scanning)
- Stopped (The program was stopped)
- Sending results (The program is sending its scanning results via email)
- Finished (The program has finished scanning and sending results)

Next to the status information there is a progress bar that indicates JavaSpy's progress while scanning. This progress bar only appears when scanning is in progress.

Next to the progress bar are four little information boxes. These boxes give information of the main settings of the current session. They are as follow:

- ML (Maximum links to scan)
- MD (Maximum depth JavaSpy searches for links)
- Proxy (Tells if connections to the Internet are made to go through a proxy)
- RES (Tells if JavaSpy is obeying the Robot Exclusion Standard)



When the Go... button is clicked, JavaSpy starts scanning a given site, the Go... button changes into a Pause button, and a Stop button appears next to it. JavaSpy can be paused while scanning or stopped. If it is stopped, a new session will have to be edited to reset JavaSpy's scanning engine. While scanning, JavaSpy displays the progress indicator as well as the links it has found in a list situated on the right hand side of the window, each link has a colour label attributed which indicates its state. There are four different colours each with a different meaning; they are as follow:

- Green (The links is fine)
- Red (A connection could not be made to this link)
- Blue (This link is external to the web site being scanned)
- Black (This link is out of range for JavaSpy)

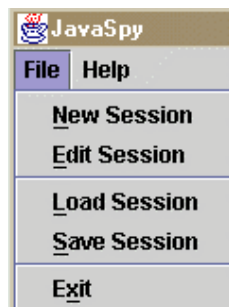
A link is external when it is not part of the current web site being scanned, for example, if the site *http://www.thesite.com* is scanned and a link found is *http://www.anothersite.com*, JavaSpy will flag this link as being external.

A link may be out of range because JavaSpy was told to go to a certain depth and ignore links below that depth, to solve this problem, increase the depth at which a scan can be done.

When a link, which could not be connected to, appears in the list, it could depend on many different things. The links may not be valid anymore or the resource it

points to does not exist any more. A login and password may be needed to open this link; JavaSpy does not yet implement logins and passwords for protected sites. The site being scanned is being updated, etc...

On the left hand side of the list, there are two areas of interest. The first area at the top give a real time indication of what JavaSpy has found so far during a scan. The second area below gives information on a chosen link, it indicates which page a link is situated, what depth it is at, if it is good or bad or external or out of range, what type of document it represents, and most important, its original form. Its original form is what the link looks like within the HTML code for that page. This makes it easier to locate it and repair it or change it if needed. To see a link's information, simply click on it in the list.

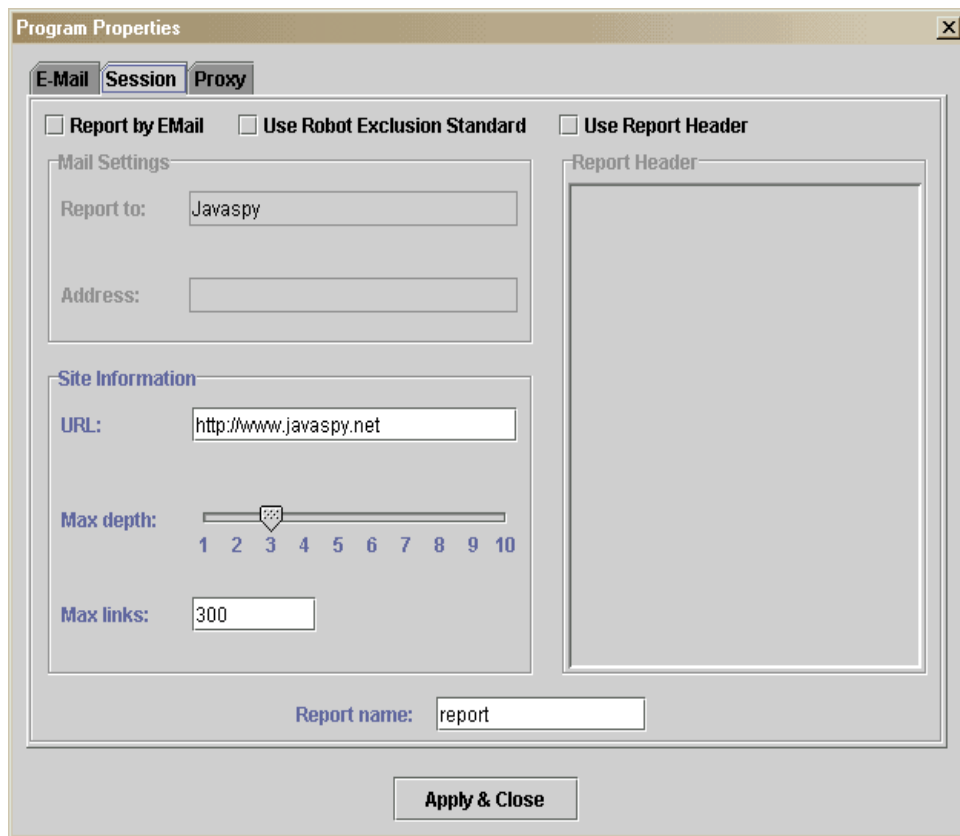


This menu allows for a new session to be made active with default settings, to edit the current session, to load a session or to save a session. There is also an exit feature that closes the program.

When new session or edit session is selected, the program properties settings dialog box appears, see the next three screen shots.

The dialog box contains three areas, they are:

- E-Mail (Change settings for email sending of scanned results)
- Session (Change the behaviour of JavaSpy)
- Proxy (Tell JavaSpy to make its connections through a proxy server)



When the program properties box is opened, the default tab selected is the session tab. In there you can tell the program which site to scan, at what depth and how many links to check for. If the Report by Email check box is selected, you can enter the address to which you wish to send the results of a scan. If the RES check box is selected, JavaSpy will obey the Robot Exclusion Standard while scanning. Results sent via email are built as an HTML document, this document has a default heading but it may be replaced by your own heading, for this, select the Use Report Header check box, and enter the desired header. Below this is a text field where the name of the report can be entered. Any name entered will be changed; here is an example.

If the name chosen is *report.txt*, the *.txt* extension will be removed, and the name will be remade to look like this: *report_260201_175834.html*. This report name means that the scan was started on the 26th February 2001 at 5:58:34 PM. This system helps in avoiding overwriting the results from a previous scan, and also gives a little more information as to when a scan was started. The filename of a report may consist of a fully qualified path name, for example:

C:\JavaSpy\reports\myreport.html, if a path name encountered does not exist, JavaSpy will create it for you and place the HTML file in this directory.

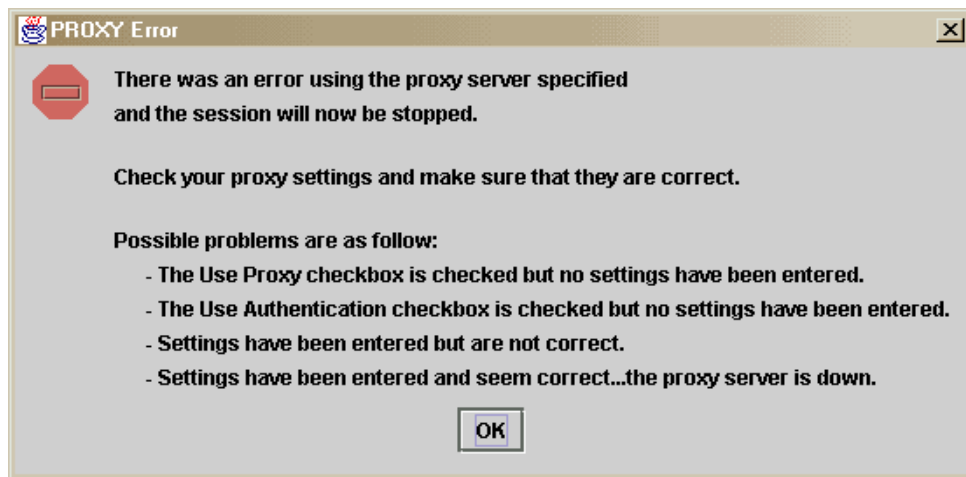
The screenshot shows a 'Program Properties' dialog box with three tabs: 'E-Mail', 'Session', and 'Proxy'. The 'E-Mail' tab is selected. It contains two sections: 'User Information' and 'Server Information'. The 'User Information' section has three text fields: 'Name' (xavier), 'Surname' (bourguignon), and 'E-Mail Address' (bg750763@stmail.staffs.ac.uk). The 'Server Information' section has a 'Protocol' section with 'POP3' selected (indicated by a filled radio button) and 'IMAP' unselected (indicated by an empty radio button). Below this are four text fields: 'Server' (stmail.staffs.ac.uk), 'Username' (bg750763), and 'Password' (masked with asterisks). At the bottom right of the dialog is an 'Apply & Close' button.

In here, you may set your email server's information. Your name, surname, email address and server name must be present, if not you will not be able to select the Report by Email check box in the session tab, therefore you will not be able to have the results sent via email. If you need a login and password to access your mail server, enter them in the fields provided. Your email server must be using the SMTP protocol in order to be able to send emails.

The image shows a screenshot of a 'Program Properties' dialog box with three tabs: 'E-Mail', 'Session', and 'Proxy'. The 'Proxy' tab is selected. Inside the 'Proxy' tab, there are two main sections. The first section, 'Proxy Server', contains two checkboxes: 'Use proxy' (which is checked) and 'Use authentication' (which is unchecked). Below these checkboxes are two text input fields: 'Address:' and 'Port:'. The second section, 'Proxy Authentication', contains two text input fields: 'Username:' and 'Password:'. At the bottom of the dialog box is a button labeled 'Apply & Close'.

In this tab, you may choose to have JavaSpy to make connections via a proxy server.

If you have selected the Use proxy check box, you must enter the address and port of your proxy, if not JavaSpy will come up with an error message, see figure 7. If your proxy server needs to authenticate you, select the Use authentication check box and enter your login and password as appropriate. Again, if you select the Use authentication check box and do not enter anything in the provided fields, JavaSpy will come up with an error message, see next page.



This error message tries to help you figuring out what could have gone wrong while connecting via a given proxy server. When the OK button is clicked, the program properties will open with the proxy tab selected to let you solve the problem. If your proxy server is down and you cannot make any connection without it, nothing can be done but wait for it to be restarted.

Choosing save or load session will open a save as or open dialog box, all sessions must be saved with the .spy extension. When loading a session, JavaSpy only displays the directories and .spy files it can see, just choose one as shown below and click open.

